NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

S8535

THE MOVING PLATFORM SIMULATOR II:
A NETWORKED REAL-TIME VISUAL SIMULATOR
WITH DISTRIBUTED PROCESSING AND
LINE-OF-SIGHT DISPLAYS

by

Randolph P. Strong and Michael C. Winn

June 1989

Thesis Advisor: Michael J. Zyda

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE

a Report Security Classification: UNCLASSIFIED	1b Restrictive Markings
a Security Classification Authority	3 Distribution Availability of Report: Approved for public release; distribution is unlimited.
b Declassification/Downgrading Schedule	
Performing Organization Report Number(s)	5 Monitoring Organization Report Number(s)
a Name of Performing Organization: Naval Postgraduate School; 6b Office Symbol (*If Applicable*): 52	7a Name of Monitoring Organization: Naval Postgraduate School
c Address (*city, state, and ZIP code*): Monterey, CA 93943-5000	7b Address (*city, state, and ZIP code*): Monterey, CA 93943-5000
a Name of Funding/Sponsoring Organization; 8b Office Symbol (*If Applicable*)	9 Procurement Instrument Identification Number
c Address (*city, state, and ZIP code*)	10 Source of Funding Numbers: Program Element Number; Project No; Task No; Work Unit Accession No

Title (*Include Security Classification*)
THE MOVING PLATFORM SIMULATOR II: A NETWORKED REAL-TIME VISUAL SIMULATOR WITH DISTRIBUTED PROCESSING AND LINE-OF-SIGHT DISPLAYS

Personal Author(s)
Randolph P. Strong and Michael C. Winn

13a Type of Report	13b Time Covered	14 Date of Report (*year, month,day*)	15 Page Count
Master's Thesis	From To	June 1989	169

Supplementary Notation
The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

Cosati Codes: eld	Group	Subgroup	18 Subject Terms (*continue on reverse if necessary and identify by block number*)
			Moving Platform Simulators, Visual Simulators, Real-Time Graphics, Distributed Processing, Line-of-Sight

Abstract (*continue on reverse if necessary and identify by block number*)
Previous research has produced a real-time Moving Platform Simulator using Defense Mapping Agency digital errain elevation data and a Silicon Graphics, Inc. Iris 4D/70GT graphics workstation. This study is a ontinuation of that effort with the multiple goals of investigating the effects on simulator performance of using igher resolution terrain and different terrain drawing algorithms. Also investigated was the integration of real-me, actual platform intervisibility determinations into the simulator. Included in this effort was a study of nodeling time and a real world coordinate system. Additional work was performed on using a distributed omputing architecture to maximize performance.

Distribution/Availability of Abstract: [X] unclassified/unlimited [ ] same as report [ ] DTIC users	21 Abstract Security Classification: UNCLASSIFIED
2a Name of Responsible Individual: Prof. Michael J. Zyda	22b Telephone (*Include Area code*): (408) 646-2305; 22c Office Symbol: Code 52Zk

D FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

Approved for public release; distribution is unlimited.

The Moving Platform Simulator II:
A Networked Real-Time Visual Simulator with
Distributed Processing and Line-of-Sight Displays

by

Randolph P. Strong
Captain, United States Army
B.S., United States Military Academy, 1978
and
Michael C. Winn
Captain, United States Marine Corps
B.S.E.E., University of Oklahoma, 1982

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1989

ABSTRACT

Previous research has produced a real-time Moving Platform Simulator using Defense Mapping Agency digital terrain elevation data and a Silicon Graphics, Inc. IRIS 4D/70GT graphics workstation. This study is a continuation of that effort with the multiple goals of investigating the effects on simulator performance of using higher resolution terrain and different terrain drawing algorithms. Also investigated was the integration of real-time, actual platform data, electronically gathered by position-location reporting instruments and platform intervisibility determinations into the simulator. Included in this effort was a study of modeling time and a real world coordinate system. Additional work was performed on using a distributed computing architecture to maximize simulator performance.

38535
C.1

TABLE OF CONTENTS

I. INTRODUCTION 1
A. BACKGROUND 1
B. CDEC REQUIREMENTS 1
C. SIMULATOR DEVELOPMENT HISTORY 2
1. FOGM MISSILE SIMULATOR 3
2. VEH VEHICLE SIMULATOR 4
3. FOGM/VEH NETWORKING SIMULATOR 4
4. VEH II VEHICLE SIMULATOR 4
5. THE MOVING PLATFORM SIMULATOR (MPS) 5
6. FOGM, VEH, VEH II, AND MPS PERFORMANCE HISTORY 6

II. MOVING PLATFORM SIMULATOR II DESCRIPTION 9
A. SYSTEM OVERVIEW 9
B. SIMULATOR UPGRADES NEEDED 9
C. DISPLAY OF REAL-TIME PLATFORMS 10
D. LINE-OF-SIGHT CALCULATION AND DISPLAY 11

III. DIGITAL TERRAIN DATABASE 12
A. BACKGROUND 12
B. LOCATION AND SIZE OF DATABASE 12
C. COORDINATE SYSTEM 13
D. STRUCTURE OF DATABASE 15
1. Coverage 15
2. Data File Format 16
3. Data Point Structure 16
E. ACCESSING THE DATABASE 18

1. Background 18
2. Database Reads 18
3. Database Access Algorithm 19
a. Initial Database Read 19
b. Subsequent Database Reads 21
4. Minimum and Maximum Elevations 21
F. RESOLUTION DECISIONS 23

IV. THE MODELING OF TIME 24
A. BACKGROUND 24
B. CAPABILITIES 24
C. LIMITATIONS 26

V. GRAPHICS DISPLAY OVERVIEW 27
A. GRAPHICS TECHNIQUES 27
1. Double Buffering 27
2. Z-buffering 27
3. RGB Color 28
4. Perspective World Views 29
a. Two-Dimensional Drawing 29
b. Three-Dimensional Drawing 29
B. TWO-DIMENSIONAL TERRAIN DISPLAYS 30
C. OVERLAYS 31
1. Thirty-five Kilometer Map 32
2. Ten Kilometer Map 32
D. THREE-DIMENSIONAL TERRAIN DISPLAYS 33
1. Background 33
2. MPS Terrain Data Structure 33
3. New Terrain Data Structure 34
4. Mesh Drawing Primitive 36
5. Distance Attenuation 37

6. Terrain Normals 40
a. Background 40
b. MPS II Terrain Normals 41
c. MPS II Vertex Normal Data Structure 42
7. The MPS Three-dimensional Terrain Display Algorithm 42
8. MPS II Three-Dimensional Terrain Display Algorithm 45
E. PLATFORM MODELING 45
1. Platform Position 45
2. Platform Orientation 47
3. Viewing Perspective 50
a. Background 50
b. Coordinate System Transformations 51

VI. NETWORKING CAPABILITIES OF MPS II 56
A. BACKGROUND 56
B. ARCHITECTURE 57
1. Protocols 58
2. Data Structures 59
a. MPS Messages 60
b. New Messages 61

VII. DISPLAY OF REAL-TIME PLATFORMS 65
A. SYSTEM ARCHITECTURE 65
1. Overview 65
2. Purpose of Modules 66
3. Interprocess Communication 68
a. Between NETWORK_SIMULATOR AND PROCESS_VDB Modules 69
(1) Protocols. 69
(2) Data Structures. 71
b. Between PROCESS_VDB and MPS II Modules 76

(1) Protocols. 76
(2) Data Structures. 76
B. THE PROCESS_VDB MODULE 79
1. Retrieving The VIDS Data Block 80
2. The VIDS Data Block Time Stamp 80
3. The VIDS Data Block Length 81
4. The VIDS Messages 82
a. The Start-Time Message 82
b. The Player Position Message 83
(1) Determination To Send Update Message 86
(2) Actions Taken When Update Must Be Sent 87
(3) Actions Taken When Update Is Not Sent 89
C. THE NETWORK_SIMULATOR MODULE 89
1. Reading the VIDS Data Block 90
2. Waiting for Elapsed Time 90
3. Releasing the VIDS Data Block 91
D. The Moving Platform Simulator II Interface 91

VIII. INTERVISIBILITY CALCULATIONS AND DISPLAY 92
A. SYSTEM ARCHITECTURE 92
1. Overview 92
2. Interprocess Communication 93
a. Protocols 94
b. Data Structures 95
B. DETERMINATION OF INTERVISIBILITY BETWEEN PLATFORMS 100
1. Optimization Rules 100
2. Determining the Elevation of a Point 103
a. Elevation Database 103
b. Elevation Calculation 104
(1) Defining The Plane. 104

(2) Finding the Equation of the Plane. 105
(3) Solving the Equation for the Unknown Elevation. . . 105
3. Target and LOS Points 107
C. PLATFORM INTERVISIBILITY DISPLAY 110
1. PROCESS_LOS 111
a. Display Layout 111
b. Display Options 112
2. MPS II 114

IX. SYSTEM EVALUATION OF MPS II 116
A. MOVING PLATFORM SIMULATION PERFORMANCE 116
B. SYSTEM LIMITATIONS 121

X. CONCLUSIONS AND FUTURE WORK 125
A. CONCLUSIONS 125
B. FUTURE WORK 126

APPENDIX A. USER INTERFACE 128
A. MPS II USER INTERFACE 128
1. STARTING THE SIMULATOR (COMMAND LINE OPTIONS) 128
2. COMMAND LINE OPTIONS 129
3. POP-UP MENU SYSTEM 130
a. Select Area Menus 131
b. Main Menus 134
c. Operating Menus 137
(1) Driving 137
(2) Flying 140
3. DIALS 141
a. Driving Dial Configuration 141
b. Flying Dial Configuration 142

4. MOUSE 142
5. KEYBOARD 145
B. NETWORK_SIMULATOR USER INTERFACE 145
C. PROCESS_VDB USER INTERFACE 145
D. PROCESS_LOS USER INTERFACE 146
1. Starting The PROCESS_LOS Module 146
2. Pop-up Menu System 147
a. Main Menu 147

LIST OF REFERENCES 153

INITIAL DISTRIBUTION LIST 155

LIST OF TABLES

TABLE 1-1 ONE VEHICLE ON TERRAIN (FRAMES/SECOND) 7

TABLE 1-2 NINE VEHICLES IN VIEW (FRAMES/SECOND) 7

TABLE 1-3 NINE VEHICLES, NONE IN VIEW (FRAMES/SECOND) . . . 7

TABLE 1-4 MPS PERFORMANCE MEASUREMENTS on an IRIS 4D/70GT 8

TABLE 3-1 VEGETATION CODES 18

TABLE 6-1 UPDATE MESSAGE BODY DEFINITION 63

TABLE 7-1 SIZE OF DATA STRUCTURES 67

TABLE 7-2 PREDEFINED VIDS MESSAGES 75

TABLE 7-3 MESSAGE BODY DEFINITIONS 78

TABLE 8-1 MESSAGE BODY DEFINITIONS 98

TABLE 9-1 MPS II PERFORMANCE MEASUREMENTS on an IRIS 4D/70GT 118

TABLE 9-2 MPS II PERFORMANCE DRAWING HIGH RESOLUTION TERRAIN 119

TABLE 9-3 MPS II PERFORMANCE DRAWING HIGH RESOLUTION TERRAIN in the NETWORK MODE 122

LIST OF FIGURES

Figure 3-1 Coordinate Systems 14

Figure 3-2 UTM Coordinate System 15

Figure 3-3 Structure of Elevation Database 17

Figure 3-4 Initial Database Read 20

Figure 3-5 Ten Square Kilometer Database Read 22

Figure 5-1 MPS Terrain Construction 35

Figure 5-2 One Point is Vertex For Six Triangles 36

Figure 5-3 MPS II Terrain Data Structure 36

Figure 5-4 Drawing With the Mesh Primitive 38

Figure 5-5 Distance Attenuation Scheme 39

Figure 5-6 Vertex Normal Computation by Cross Product 43

Figure 5-7 Determining Field of View 44

Figure 5-8 Determining Viewing Direction 46

Figure 5-9 Mesh Drawing Routine 47

Figure 5-10 Updating Platform's Position 48

Figure 5-11 Eye Position of MPS 52

Figure 5-12 Corrected Eye Position of MPS II 53

Figure 5-13 Transforming Platform Coordinates to World Coordinates 54

Figure 5-14 Correct Look-at Point From Driven Platform 55

Figure 6-1 Interprocess Communication Links 58

Figure 6-2 Example Update Message 63
Figure 7-1 Interprocess Communication Links 70
Figure 7-2 VIDS Data Block 73
Figure 7-3 VIDS Data Block Header Structure 74
Figure 7-4 VIDS Messages Structure 74
Figure 7-5 Example Messages 77
Figure 7-6 Algorithm to Locate Start of VIDS Data Block 81
Figure 7-7 Start Time Message Structure 83
Figure 7-8 Player Position Message Structure 85
Figure 8-1 Interprocess Communications Links 94
Figure 8-2 Example Messages 97
Figure 8-3 Intervisibility Algorithm 101
Figure 8-4 Normalization Algorithm 104
Figure 8-5 Selection of Database Points Used to Calculate the Elevation of a Position Not Represented in the Database 106
Figure 8-6 Algorithm to Calculate Elevation of Point in Upper Left Triangle 107
Figure 8-7 Algorithm to Calculate Elevation of Point in Lower Right Triangle 108
Figure 8-8 Side View of Target and LOS Points 109
Figure 8-9 Overhead View of Target and LOS Points 110
Figure 8-10 Window Layout 111

Figure 8-11 PROCESS_LOS Display of 10 x 10 Km Map With Line-Of-Sight Trace 113
Figure 8-12 PROCESS_LOS Display of 10 x 10 Km Map Without Line-Of-Sight Trace 115
Figure 9-1 Performance Measurement Comparisons 120
Figure 9-2 Terrain Display Using 100 Meter Resolution 123
Figure 9-3 Terrain Display Using 12.5 Meter Resolution 123
Figure A-1 Dial Box With Dials Labeled For Driving 143
Figure A-2 Dial Box With Dials Labeled For Flying 144

I. INTRODUCTION

A. BACKGROUND

Previous research at the Naval Postgraduate School has developed a real-time Moving Platform Simulator (MPS) with vehicles both driving and flying over three-dimensional, digitally-derived terrain [Ref. 1]. That simulator was developed on a Silicon Graphics, Inc. IRIS 4D/70GT high-performance graphics workstation. This study continues that work with the goal of implementing a simulator that displays actual vehicles in real time as they maneuver in a war gaming situation. One major goal is to display realistic three-dimensional terrain and neighboring vehicles as they would actually be seen from a driven vehicle. In addition, real-time tactical intervisibility statistics are computed and displayed. Such a visual simulator is needed by the United States Army Combat Developments Experimentation Center (CDEC), Fort Ord, California, for use in evaluating new weapons systems for use by the United States armed forces. This work is also the continuation of ongoing research into meaningful performance measurements of real-time graphics workstations [Ref. 1].

B. CDEC REQUIREMENTS

CDEC conducts its major experiments at Fort Hunter-Liggett (FHL), California. With over 1200 square kilometers of navigable terrain, FHL provides an ideal location to conduct experiments with moving vehicles. Employing a sophisticated array of instrumentation equipment, CDEC is able to track and record the movements of the

vehicles across the terrain. The data include vehicle position information at an instance in time sent by transmitters located on the vehicles and received by equipment at the FHL cantonment area. It is this data that we are interested in graphically displaying. CDEC is presently using an earlier version of the simulator and has requested support in upgrading for present and future needs.

To be useful, the simulator should be able to generate realistic high-resolution terrain in real time, with no artificial boundaries placed on the movement of vehicles about the terrain. Provided with the vehicle location data, the simulator should process and display these vehicles in their proper locations in real time. The capability of recording these scenarios for playback is also desired. This would give the simulator the capability of replaying an exercise so that evaluators could better judge the performance of a weapon system or its operator. The ability to add fictional vehicles for "what if" situations was also desired. CDEC also requested that selected intervisibility (line-of-sight) computations between designated positions be displayed in a two-dimensional graphical display at the completion of any segment of an exercise.

C. SIMULATOR DEVELOPMENT HISTORY

MPS II has evolved from a number of previous students' endeavors at the Naval Postgraduate School. In order to better understand the Moving Platform Simulator II (MPS II) and the modifications made to the Moving Platform Simulator (MPS), we need to discuss the systems from which it evolved.

1. FOGM MISSILE SIMULATOR

The Fiber-Optically Guided Missile (FOGM) simulator [Ref. 2] was developed on a Silicon Graphics, Inc. IRIS 3120 graphics workstation. The FOGM simulator presented the user with a three-dimensional image simulating the view from the nose camera of a FOGM missile. The missile flew over a fixed ten square kilometer area of Fort Hunter-Liggett, California, and was able to target, track, and destroy ground vehicles. The terrain elevation data used to depict the terrain was an extract of a specially generated database provided by CDEC. That database is discussed in detail in Chapter III.

For this early simulation, the most arduous task was providing for real-time, hidden-surface elimination. The IRIS 3120 has no special-purpose hardware to support real-time, double-buffered, hidden-surface elimination. The authors instead used a scanline Painter's algorithm [Ref. 3]. The Painter's algorithm sorts all polygons to be drawn, ensuring the polygons farthest away from the viewer's eye are drawn first. This guarantees that objects closer to the viewer are not obscured by objects further away. Care was also taken to ensure that vehicles were drawn on top of the terrain, oriented correctly, and in the proper grid location. Vehicles had to be drawn after the terrain on which they appeared was drawn, and special consideration had to be given for vehicles near the boundary of two grid squares. Vehicles were given initial speeds and headings, but the operator had no control beyond this.

2. VEH VEHICLE SIMULATOR

The VEH vehicle simulator [Ref. 3] was also developed on the Silicon Graphics, Inc. IRIS 3120 graphics workstation. It uses similar algorithms as the FOGM simulator for terrain and vehicle display, but only the terrain in the field of view is displayed. The VEH simulator also offered the operator control to maneuver a vehicle as it traversed the terrain.

3. FOGM/VEH NETWORKING SIMULATOR

The FOGM and VEH simulators were combined via the Ethernet local area network to form the FOGM/VEH NET simulator. This combination allowed a vehicle being driven on one simulator (VEH) to be displayed on another workstation (FOGM). Similarly, the FOGM missile could be flown on one workstation and displayed on another.

4. VEH II VEHICLE SIMULATOR

The VEH simulator was modified and enhanced to form the vehicle simulator, VEH II. Improvements to the VEH include the following:

- Conversion for operation under the MEX window management system on the IRIS 4D/70G [Ref. 4] graphics workstation.
- Pop-up menu user interface for user-controlled selection.
- Additional vehicles could be added after operation had begun.
- Pre-saved vehicle convoy files could be selected from pop-up menus.
- Current convoys could be saved to a file for future use.
- Multiple processes could be present using the window management system.

VEH II was later modified to operate on the Silicon Graphics 4D/70GT high-performance graphics workstation under the 4Sight [Ref. 5] window management system. The functionality of the simulator did not change substantially.

5. THE MOVING PLATFORM SIMULATOR (MPS)

The Moving Platform Simulator (MPS) [Ref. 1] is a combination of the FOGM and VEH II simulators. MPS was specifically designed to take advantage of the specialized graphics hardware of the Silicon Graphics IRIS 4D/70GT high-performance graphics workstation. The idea was to test manufacturers' claims of graphics performance in an actual applications environment. Some of MPS's enhanced capabilities include the following:

- Complete integration under the 4Sight [Ref.5] window management system.
- Selectable ten square kilometer operations area from a 35 square kilometer database.
- Multiple terrain elevation color schemes.
- Realistic lighting model for shading of terrain and vehicles that includes user-selected month and hour.
- An improved terrain-drawing algorithm that uses distance attenuation for improved performance.
- Hidden surface removal using Z-buffering.
- Vehicle collision detection.
- FOGM's ability to track and destroy land vehicles.
- Broadcast networking over the Ethernet local area network for simultaneous multi-workstation simulations.

Since MPS II is a modification of MPS, it is important to understand the capabilities and limitations of MPS in greater detail to better understand the modification and improvements made during the development of MPS II. Therefore, subsequent chapters go into more detail about the functionality of MPS as the modifications which constitute MPS II are discussed.

6. FOGM, VEH, VEH II, AND MPS PERFORMANCE HISTORY

When comparing performance statistics among graphics simulators, a common unit of measurement is the number of frames per second drawn, i.e., the number of times per second an entire scene can be redrawn and displayed. The FOGM and VEH simulators operated on the relatively slow central processing unit (CPU) of the IRIS 3120, and achieved only six to eight frames per second. After being ported to the IRIS 4D/70G, the simulators achieved almost twice the speed as before. This speed increase can be attributed to the faster ten million instruction per second (MIPS) CPU.

Although the IRIS 4D/70GT also operates with a 10 MIPS CPU, it contains improved special purpose graphics hardware. The VEH II obtained over twice as many frames per second as its predecessor. See Table 1-1, Table 1-2 and Table 1-3 for actual test results comparing the different processors and simulators.

The Moving Platform Simulator is a much more sophisticated simulator, and cannot be compared directly with either the VEH or VEH II simulators. Many new options were added to take advantage of new and improved hardware, but some of these improvements degraded its performance. Most noticeable is the ability to draw terrain to the edge of a ten kilometer sector. Earlier simulators displayed only 2000 meters of terrain in front of the driven vehicle. Even though a distance attenuation scheme is used, this additional display load decreases the system's performance. Since MPS is the first of a new class of simulators, a benchmark needed to be established to judge its performance [Ref. 1]. Quantities of polygons per frame and frames per

second measurements were used. See Table 1-4 for the results of performance testing of MPS. These figures are used as a benchmark to judge the performance of MPS II.

TABLE 1-1 ONE VEHICLE ON TERRAIN (FRAMES/SECOND)

SIMULATOR/MACHINE	15 DEGREE VIEW	55 DEGREE VIEW
VEH/3120	8.0	6.0
VEH II-4D/70G	14.0	7.0
VEH II-4D/70GT	30.0	16.0

TABLE 1-2 NINE VEHICLES IN VIEW (FRAMES/SECOND)

SIMULATOR/MACHINE	15 DEGREE VIEW	55 DEGREE VIEW
VEH/3120	4.0	3.5
VEH II-4D/70G	5.0	3.0
VEH II-4D/70GT	10.0	6.0

TABLE 1-3 NINE VEHICLES, NONE IN VIEW (FRAMES/SECOND)

SIMULATOR/MACHINE	15 DEGREE VIEW	55 DEGREE VIEW
VEH/3120	6.0	5.0
VEH II-4D/70G	12.0	7.0
VEH II-4D/70GT	25.0	16.0

TABLE 1-4 MPS PERFORMANCE MEASUREMENTS on an IRIS 4D/70GT

DISPLAYING DETAILED TERRAIN

PLATFORM	ZOOM ANGLE (DEGREES)	POLYGONS PER FRAME	FRAMES PER SECOND
ONE VEHICLE	55	763	8
ONE VEHICLE	15	403	14
NINE VEHICLES	55	1086	6
NINE VEHICLES	15	722	8
MISSILE 1500m	90	19801	< 1
MISSILE 1500m	10	3387	2

DISPLAYING ATTENUATED TERRAIN

PLATFORM	ZOOM ANGLE (DEGREES)	POLYGONS PER FRAME	FRAMES PER SECOND
ONE VEHICLE	55	607	9
ONE VEHICLE	15	393	15
NINE VEHICLES	55	940	7
NINE VEHICLES	15	680	9
MISSILE 1500m	90	4152	2
MISSILE 1500m	10	816	7

II. MOVING PLATFORM SIMULATOR II DESCRIPTION

A. SYSTEM OVERVIEW

The Moving Platform Simulator II is an enhancement of MPS's capabilities to better suit the specific needs of CDEC, while at the same time pursuing the ongoing research into the graphics capabilities of high-performance graphics workstations. For this reason, MPS II is in many ways similar to its predecessor. Enhancements made were either necessary to achieve the desired functionality or to explore a specific topic of interest. With a project of this magnitude, there are always many improvements that can be made, but because of limited time, only those changes deemed as important steps toward reaching the goals of the project were implemented.

B. SIMULATOR UPGRADES NEEDED

The initial enhancement needed was the capability of displaying user-selected, multiple-resolution terrain in both two and three dimensions. The program's dependencies upon the resolution of the digital terrain database had to be removed and the drawing algorithm revised to allow the real-time display of higher-resolution terrain. The drawing routines also had to be modified so that the terrain display looked more realistic and true to life.

Secondly, the location of vehicles being displayed had to be converted to the Universal Transverse Mercator (UTM) projection system, the standard grid coordinate system used by the United States military land forces. This coordinate system upgrade

was needed to eliminate the artificial boundaries placed on the movement of vehicles by the MPS system.

The next important revision needed was the implementation of a real-time simulator clock. The reason this was important is that MPS II is required to operate in a real-time mode receiving platform update information about the actual vehicles. The simulator cannot be allowed to stop the simulator time clock while it carries out functions such as reading from disk, executing pop-up menus, or displaying terrain.

C. DISPLAY OF REAL-TIME PLATFORMS

To move the raw data from the instrumentation equipment to processors, and to move the data from processor to processor, CDEC uses an Ethernet local area network (LAN) and TCP/IP network protocols at FHL. During a live exercise, this network provides an ideal conduit for passing and obtaining information needed to display the vehicles in real time. During non-exercise simulations, the data in the network must be artificially generated and inserted into the network. It is at this point in this area that our work begins. To operate the simulator during non-exercise conditions, we must first load the network with data identical to the data available during an exercise. This raw data must then be retrieved from the network, processed, and converted into a format that can be handled by the Moving Platform Simulator. Since CDEC also desires the capability of displaying line-of-sight (LOS) information, the raw data must also be converted to a format that can be handled by a process that performs the LOS task.

D. LINE-OF-SIGHT CALCULATION AND DISPLAY

In addition to using the Moving Platform Simulator as described above, to successfully perform an experiment in the near future, CDEC needs the capability to display LOS information displayed on a two-dimensional map of FHL. Given the position of an observer, the program needs to graphically display the location of the observer and the path of vehicles moving over the terrain. The path of the vehicles should be color coded to indicate when the observer can and cannot visually see the vehicle. The LOS display needs to be an integral part of the Moving Platform Simulator, but not place a burden on the performance of the simulator during its operation. Since the calculation of LOS information is mathematically intensive and requires considerable CPU time, the only way to perform the calculations without degrading the performance of the simulator is to do the calculations on another processor and send only simple drawing instructions to the Moving Platform Simulator.

III. DIGITAL TERRAIN DATABASE

A. BACKGROUND

The digitally-derived terrain database used by MPS II was provided to the Naval Postgraduate School by CDEC. It is a special Defense Mapping Agency (DMA) digital terrain elevation database (DTED) file. It was produced in 1980 for the area which includes Fort Hunter-Liggett (FHL), California. A standard Level 1 DTED file [Ref. 6] contains data points spaced approximately every 100 meters. This specially-generated database contains data points spaced at 12.5 meter intervals. Thus, the resolution of this database is eight times that of a standard Level 1 DTED data file.

B. LOCATION AND SIZE OF DATABASE

The database is presently stored locally on the disk drives of two IRIS 4D/70GT's located in the Naval Postgraduate School's Graphics and Video Laboratory. The file must be accessed each time the simulator is started. The current file can be found under the path name: ~cdec/DTED/terrain.dat. This path name is referenced in the header file titled "files.h", and should the database ever be moved to a different directory, this header file must be updated to reflect the change.

As can be quickly calculated using Equation 3-1, a coverage of 36 x 35 kilometers with a 12.5 meter data point interval yields a database size of 16,128,000 bytes.

dBase_Size = pts_per_km * num_sqr_km * bytes_per_sample

where: $pts\_per\_km^2 = (8 * 10)^2 = 6400;$
$num\_sqr\_km = 35 * 36 = 1260;$
$bytes\_per\_sample = 2;$

Equation 3-1

C. COORDINATE SYSTEM

The graphics software library provided by Silicon Graphics, Inc. for use on the IRIS 4D/70GT uses a right-handed coordinate system (see Figure 3-1). The Z-axis measures distance perpendicular to the plane of the display screen, with the negative Z-axis going into the screen.

Another coordinate system used in military maps is called the Universal Transverse Mercator (UTM) projection (see Figure 3-2). In the UTM system, points are represented as a distance in meters North (northing) and East (easting) from a Grid Zone origin. The Grid Zone is labeled by a Grid Zone Designator.

One of the most important modifications needed to allow the display of actual vehicles on the terrain in real time was the integration of the UTM grid coordinate system into MPS II. These changes involved allowing vehicles to move anywhere within the area of operation and not be limited to the ten square kilometer display area. Originally in MPS, vehicles were not allowed to leave a ten kilometer area; if the user changed areas, the vehicle appeared in the same relative location in the new area.

To make these changes, new fields had to be added to the vehicle data structure. Also, a set of conversion routines had to be written to convert between UTM coordinates and the world and screen coordinates used in the drawing routines. Care had to be used to only update and draw vehicles in the ten kilometer area since

elevation data for areas outside this area is not in memory. These topics are discussed fully in Chapter V.

Work has just been completed on the FOST system (a variant of MPS) to read any Level 1 DTED file, and convert its system of longitudes and latitudes to UTM coordinates [Ref. 7]. MPS II cannot yet read arbitrary DTED level 1 files, but will a later date.

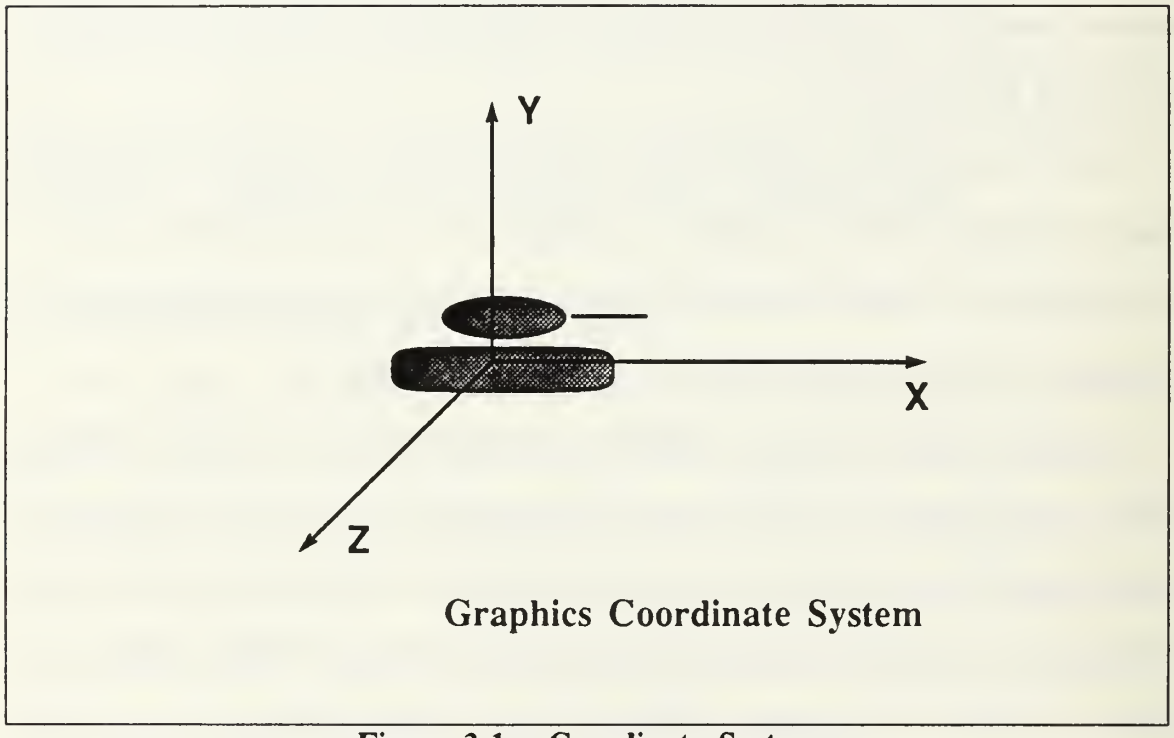


Figure 3-1 Coordinate Systems

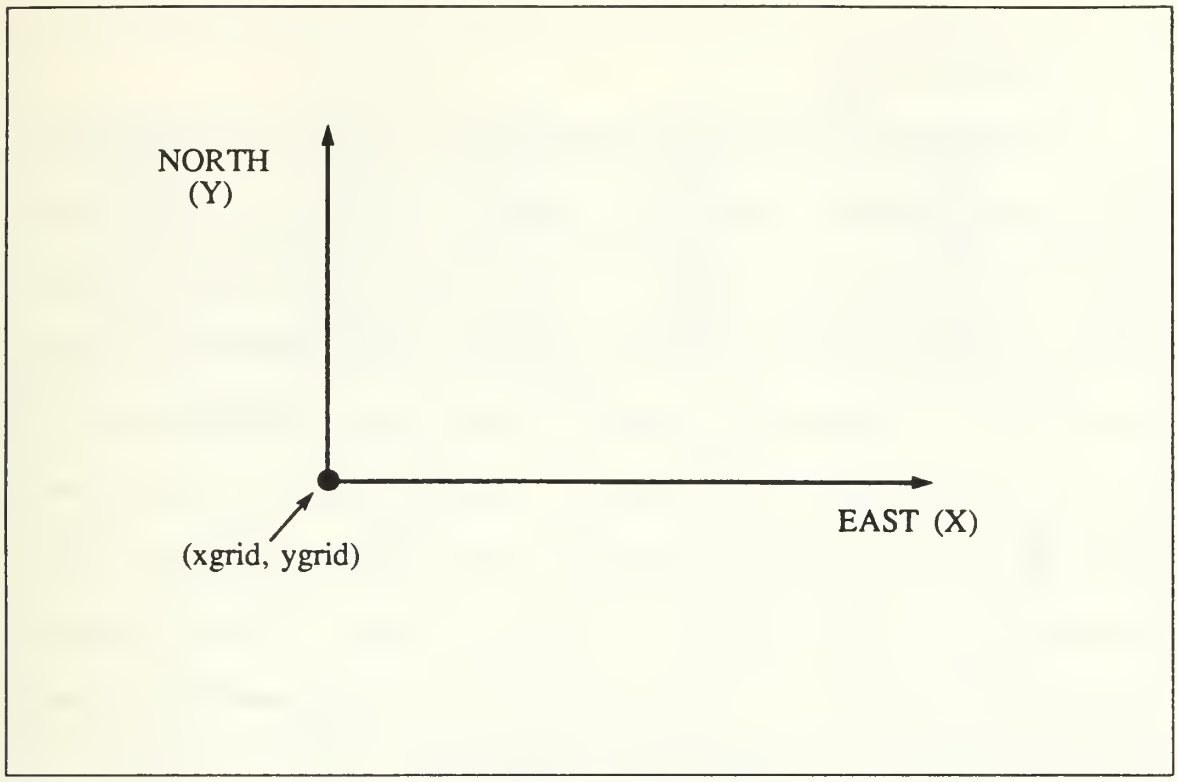


Figure 3-2 UTM Coordinate System

D. STRUCTURE OF DATABASE

1. Coverage

As mentioned above, the database file encompasses an area of 36 kilometers by 35 kilometers, which includes the area of FHL and some of the surrounding areas. It is bounded by latitude 36 degrees 5 minutes to the north, and by latitude 35 degrees 50 minutes on the south. To the west, its boundary is longitude 121 degrees 20 minutes 30 seconds, and the east boundary is 121 degrees, 4 minutes, 30 seconds. These boundaries convert to Universal Transverse Mercator (UTM) coordinates as follows:

Eastings from 10SFQ41000 to 10SFQ77000

Northings from 10SFQ60000 to 10SFQ95000

2. Data File Format

The file is stored sequentially and unformatted as a stream of 16 bit integers, each two bytes representing a single elevation point. The order in which the data points are stored is illustrated in Figure 3-3. The database begins with the extreme southwest corner corresponding to UTM grid coordinate 10SFQ6000095000. The data points of each one kilometer grid square are grouped and stored sequentially in columns, starting with the lower left hand point and proceeding up the column one kilometer (80 data points), then proceeding to the next column from bottom to top, repeating this pattern for 80 columns (one kilometer of columns). These grid squares are arranged in a similar fashion, in columns starting with the southwestern most square and proceeding from south to north, and from west to east.

3. Data Point Structure

As mentioned above, each data point is represented by a 16 bit integer. Within this integer are stored both an elevation code and a vegetation code. The 13 least significant bits represent elevation in feet, not including the height of vegetation. Employing thirteen bits, elevations ranging from zero to 8191 feet can be represented. The three most significant bits represent one of eight vegetation codes. These vegetation codes are only available for the area within Fort Hunter-Liggett proper, and for this reason, are not used in MPS II. The vegetation codes are listed in Table 3-1.

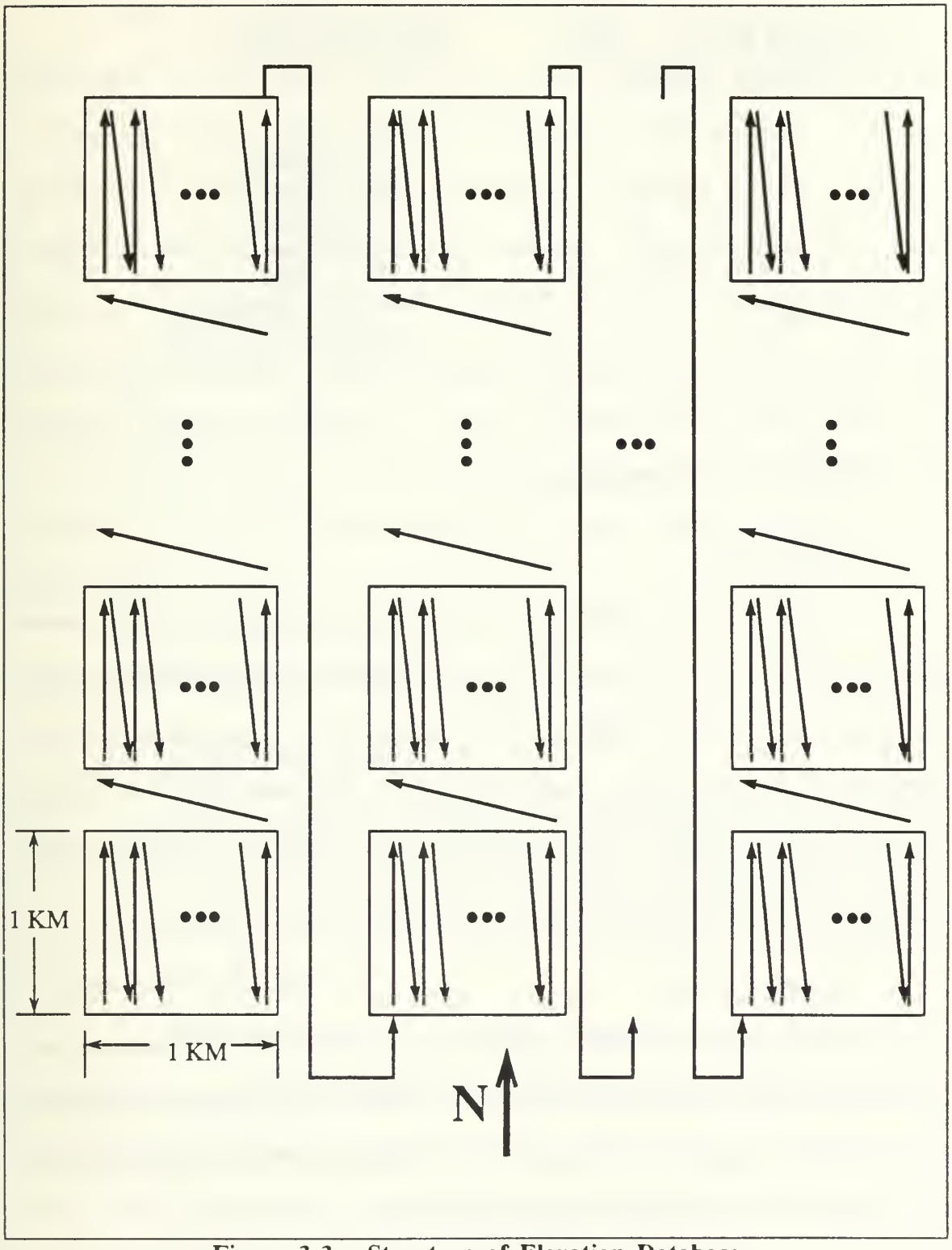


Figure 3-3 Structure of Elevation Database

TABLE 3-1 VEGETATION CODES

VEGETATION CODE		DESCRIPTION
binary	decimal	
000	0	<1 Meter
001	1	1 - 4 Meters
010	2	4 - 8 Meters
011	3	8 - 12 Meters
100	4	12 - 20 Meters
101	5	>20 Meters
110	6	No Data Available
111	7	Not Used at This Time

E. ACCESSING THE DATABASE

1. Background

Because the original MPS was not concerned with displaying high-resolution terrain, its authors extracted every eighth point (100 meter resolution) of the original terrain database for use with MPS. This database subset was restructured into two-dimensional array format which made accessing the data much faster and simpler. Since a major goal of MPS II is to demonstrate the capability of displaying high-resolution terrain, the terrain data file in its original fonn was used.

2. Database Reads

The database is accessed on three distinct occasions by the simulator and LOS module. Both the simulator and the LOS module read the entire 16 Megabyte file at start-up. Because of the size of the database and its structure, there is approximately a 30 second delay while reading the file.

The LOS module restructures the data into a two-dimensional array in internal memory and, therefore, always has the entire database in memory throughout the simulation. Because of limitations on the amount of internal memory and the size of other data structures used in MPS II, the entire 16 Megabyte database is not held in internal memory during operation of the simulator. During the initial read of the database, every eighth point (100 meter resolution) is saved in a two-dimensional array and the rest is discarded. Greater resolution is not needed for the initial two-dimensional display (see Chapter V). After a ten square kilometer area of operation has been selected, there is another disk access. During this read, the simulator restructures the data into a two-dimensional array and saves the full 12.5 meter resolution data.

3. Database Access Algorithm

a. Initial Database Read

After the simulator initialization routines, the initial read of the database occurs. A wait bar is displayed during this read. The code which accomplishes this task is located in routine *display_big_map.c*, and the algorithm used to execute this read is presented in Figure 3-4.

Notice the routine involves four imbedded "for" loops. The database is 36 kilometers wide, but MPS II only uses the first 35 kilometers. Taking advantage of the structure of the data base, MPS II completes 35 block reads of the data file. A block of data one kilometer wide and 35 kilometers high is read into memory. The outer loop accomplishes this task, saving the data in the linear array *in_elev*. The three

```
display_big_map()
{
   short  data[Z_BIG_DATA_PTS][X_BIG_DATA_PTS];
   short  in_elev[BYTES_PER_COL];
   int    fd,count;
   short  gridcol,gridrow,col,row;

   fd = open(TERRAIN_DATAFILE,RD); /* Open data file to read.*/
   lseek(fd,0,0);  /* Move file pointer to beginning of file. */

   for (gridcol = 0; gridcol < THIRTYFIVE_SQUARES; gridcol++)
   { /* Loop once for each column for 35 columns.  */
      count = 0;
      read(fd, &in_elev[0], BYTES_PER_COL); /* Read entire column,
                                                1 x 35 km.*/
      for (gridrow= 0; gridrow < THIRTYFIVE_SQUARES; gridrow++)
      { /* Loop once for each of 35 grid squares in the column.*/

         /* Restructure the column of squares in 2-D array.*/
         for ( col = gridcol * PTS_PER_K;
                  col < (gridcol * PTS_PER_K) + PTS_PER_K; col++)
         {

            for ( row = gridrow * PTS_PER_K;
                     row < (gridrow * PTS_PER_K) + PTS_PER_K; row++)
            {
               data[col][row] = in_elev[count];
               count+=PTS_PER_HUNDRED;  /* Increment count by 8.*/
            }
            count += PTS_BETWEEN_COL; /* Increment counter to first
                                         point in next column.*/
         }
      }
   }
}
```

Figure 3-4 Initial Database Read

inside loops restructure this array into a two-dimensional array *data*. Of note is that *in_elev* is indexed by the counter variable *count*, which is incremented by a value of eight. This has the effect of skipping to every eighth point in the array which reduces

the resolution of the database to 100 meters. *In_elev* is then overwritten by the next block read.

b. Subsequent Database Reads

After a ten kilometer area of operation has been selected, the database is again accessed. The algorithm used to read, restructure, and store the high-resolution elevation data is shown in Figure 3-5. This routine is found in file "read_data.c", and is similar to the above read operation with two major exceptions. First, an offset to the first data point of each column must be calculated, and second, full resolution data is saved in an array named *dted*. This array is used by subsequent display routines.

4. Minimum and Maximum Elevations

In the original MPS, the minimum and maximum elevations were constant and absolute. The range of elevation formed from them was divided into eight intervals, and a color was assigned to each elevation interval. This method is simple, yet has the disadvantage that in an area of operation which is flat, there are very few colors displayed; thus, it is harder to discern the elevation changes. To improve on this situation, MPS II does two things differently. First, as the data is being read, minimum and maximum elevations are saved. These minimum and maximum values obviously are different depending upon the area of operation chosen by the user. Secondly, this range of elevations is divided into 16 groups (instead of eight), each of which is assigned a color. This gives a greater contrast of colors across the range of

```
#define  MAX_X_POINTS    801
#define  TEN_SQUARES     10
#define  SIZE_COL        128000

read_data()
{
  short dted[801][801];
  short x_grid, y_grid, resolution, minelev, maxelev;
  int offset, count, fd;
  short row,col,gridcol,gridrow,initx,initz;
  short in_elev[SIZE_COL];

     /* Open the file with the 12.5 meter data points in it */
     fd = open(TERRAIN_DATAFILE,RD);
     initx = (x_grid - LOWER_LEFT_X)/10;/* Calculate the lower left */
     initz = (y_grid - LOWER_LEFT_Y)/10;/*starting point in the file. */

     offset = 2*((initx*PTS_PER_GRID*GRIDS_PER_COL) +
                          (initz*PTS_PER_GRID));
     lseek(fd, offset ,0);

     for (gridcol=0; gridcol<TEN_SQUARES; gridcol++)
     {
        count = 0;
        read(fd, &in_elev[0], SIZE_COL);

        for (gridrow= 0; gridrow < TEN_SQUARES; gridrow++)
           for ( col=gridcol*PTS_ON_SIDE;
                 col<(gridcol*PTS_ON_SIDE)+PTS_ON_SIDE; col++)
              for ( row=gridrow*PTS_ON_SIDE;
                    row<(gridrow*PTS_ON_SIDE)+PTS_ON_SIDE; row++)
              {
                 dted[col][row] = ( (in_elev[count] & 0x1fff) *
                                                  FEET_TO_METERS) ;
                 count++;
              }
           lseek(fd,(25*PTS_PER_GRID*BYTES_PER_POINT),1);
        close(fd);
}
```

Figure 3-5 Ten Square Kilometer Database Read

elevations and a more accurate display of the elevation changes. The different color schemes available remain the same as in the original MPS [Ref. 1].

F. RESOLUTION DECISIONS

One of the key issues being investigated during this project was the question of what terrain resolution present-day technology computers could display in real time and still give some semblance of smooth motion. To best investigate this question, it was desirable that the user be able to select the resolution of the terrain on the fly and be able to do performance measurements. It was also desired to make the code independent of the resolution of the database.

Since the resolution of the CDEC database was 12.5 meters, and since standard Level 1 DTED files are 100 meter resolution, it was decided to make the options of resolutions for MPS II be multiples of 12.5 up to 100, (i.e., 12.5, 25, 50, 75, 100). This decision makes the code relatively dependent upon a 12.5 meter database, but it was decided for testing purposes this was satisfactory. Future versions should handle this question differently (see Chapter XI).

IV. THE MODELING OF TIME

A. BACKGROUND

The original MPS does not attempt to model time. First, platforms can move only in the three-dimensional mode. Second, all platforms are halted during pop-up menus. When two or more copies of MPS are running in the networking mode, if any one simulator enters a pop-up menu, a message is sent to all simulators on the network. This message temporarily halts all platforms for that simulator. A subsequent message is sent to restart the platforms once the simulator is out of the pop-up menu.

B. CAPABILITIES

The mission of MPS II is to display real-time platforms and, therefore, time needed to be modeled so that platform positions could continuously move. To accomplish this, a very simple model of time was chosen. There is no attempt to keep track of actual time since this would have required a set of synchronizing routines. Instead, MPS II works entirely with elapsed time, using the CPU system clock as a reference.

The MPS II module is implemented with three main operating states:

- 35 kilometer two-dimensional display
- 10 kilometer two-dimensional display
- 10 kilometer three-dimension display

The 35 kilometer and 10 kilometer two-dimensional states were modified to execute in a loop to allow continuous movement with the passing of time. The three-dimensional state was already designed as a drawing loop. Every platform's position is updated, and the network is checked for incoming platform messages once during each loop. This timing model allows MPS II to display moving vehicles in both the two and three-dimensional modes. These displays are discussed further in Chapter V.

When implementing this model, it was found that care had to be taken updating the positions of non-local platforms (network and real-time platforms). During each loop iteration, incoming messages are received and processed. These messages update the position of some non-local platforms. After all the incoming messages are processed, the function *update_veh_pos*() is called to advance the position of all (both local and non-local) platforms relative to their last known position by adding their movement during the elapsed time. However, a problem occurs with those non-local platforms that were just updated via the network. They would have been updated a second time by the function *update_veh_pos()*. This second update would incorrectly position the platform. Since the reported position of these newly updated non-local platforms is their actual position in time, there is no need to advance their movement based on the elapsed time. The platform would have been correctly positioned by the network, however, incorrectly positioned by the function *update_veh_pos()*. The resulting display showed the platform's position jumping every time an update packet was received.

The solution to the problem was to add an UPDATED flag field to each platform's data record. This flag was set to TRUE anytime an update packet was

received and the flag was reset FALSE during the following update routine. On the ensuing iteration of the loop, the platform is again updated as normal. In other words, any platform whose UPDATED flag is TRUE, is not updated. The flag stays TRUE for only one iteration of the drawing loop at a time.

This model of time allows MPS II to display moving icons, representing platforms, over the two-dimensional maps. To accomplish this, the program had to be modified so that pop-up menus do not automatically appear. Instead, the user is prompted to press the right mouse button for a menu. The workstation interface software is designed such that when a pop-up menu is called, the calling process halts until a selection has been made. As a result, all display movement stops during pop-up menus. After the menu selection has been made and the selection handled, the elapsed time is used to update the position of all the platforms. Care must be taken by the user not to stay in pop-up menus for long periods without making a selection. In such a case, the network input buffer overflows causing incoming messages to be lost. This situation causes no problem during normal operation of the simulator as vehicle positions not updated from the net are updated via "dead-reckoning".

C. LIMITATIONS

This type of model does have limitations. There is no time-of-day capability to allow for time-oriented events. Additionally, the time is not synchronized across the network. Each simulator keeps time independent of the other simulators. A single system-wide clock is a logical step in modeling time, but was out of the scope of this work.

V. GRAPHICS DISPLAY OVERVIEW

A. GRAPHICS TECHNIQUES

The Moving Platform Simulator uses many graphics techniques throughout the simulator and the LOS module. These graphics techniques are designed to take advantage of the special-purpose hardware possessed by the IRIS 4D/70GT, and to help give MPS II the speed needed to perform the desired drawing routines.

1. Double Buffering

Double buffering is a technique which, as the name implies, uses two memory buffers for the display screen. Only one buffer contains the picture currently being displayed (the front buffer), while drawing takes place in the other buffer (the back buffer). When the frame is completed in the back buffer, a call to the function *swapbuffers*() displays the new frame. The next frame is then drawn in the back buffer. Using this technique, the viewer sees only completed frames. Without this technique, the appearance of smooth motion is impossible.

2. Z-buffering

Z-buffering [Ref. 8] is a general hidden-surface elimination technique. As mentioned above, the earlier simulators developed at the Naval Postgraduate School used the scanline Painter's algorithm. The original MPS was the first to take advantage of the IRIS 4D/70GT's special-purpose hardware which accomplishes Z-

buffering with no time penalty. Earlier versions of the hardware could not use double buffering and Z-buffering simultaneously.

The concept of Z-buffering is quite simple, and the special-purpose hardware makes it easy, fast, and efficient to execute. As shown in Figure 3-2, the coordinate system is structured such that the Z-axis is perpendicular to the screen. The system allocates 24 bits of memory to the Z-buffer for every screen pixel. Before drawing a particular frame, the Z-buffer is initialized with the largest value possible, ($7fffff_{hex}$ for 24 bits). As the polygons comprising the picture are displayed, the system keeps track of the z-coordinate of the polygon for each pixel. If this value is less than the current Z-buffer value for that particular pixel, the z-coordinate is stored in the Z-buffer, and the correct color value is stored for that pixel in the display buffer. Otherwise, the color value for that polygon is not placed in the display buffer. Therefore, only the polygons with the closest z-value at each pixel are drawn, and hidden-surface elimination is accomplished.

3. RGB Color

The IRIS 4D/70GT has two color modes. The first, called the Color Map mode, is primarily included for compatibility with earlier software. Using this method, the red, green, and blue color values for a particular desired color are saved in a color lookup table (map), and selected via a particular index assigned to it. This was the method used by both the FOGM and VEH simulators.

The second color mode available is called the RGB mode, and is invoked with the call *RGBmode*(). Thereafter, a particular color is set with a call to *RGBcolor*(r, g, b) where the parameters r, g, and b are the amount of red, green, and

blue component desired in the color[1]. The original MPS had to use this color method because the Color Map Mode does not work with the real-time lighting and shading as implemented by MPS [Ref. 1,p. 19]. The lighting and shading model used by MPS II is discussed later in this chapter.

4. Perspective World Views

a. Two-Dimensional Drawing

When drawing on the IRIS 4D/70GT, a world coordinate system for each window is defined. For two-dimensional drawing, the coordinates are set using the function called *ortho2()*. The values chosen for these coordinates are chosen for the convenience of the programmer. In many cases, the coordinate systems used by MPS II are those used by MPS; however, if significant changes had to be made in a drawing routine, the coordinates were changed to better suit the drawing scheme. For example, when drawing the two-dimensional maps, the UTM boundaries of the area drawn were used; therefore, no conversion from UTM to world coordinates was needed. This also simplifies the drawing routines.

b. Three-Dimensional Drawing

To give the three-dimensional drawings a realistic appearance, the functions *perspective*() and *lookat*() are used. These functions control features such as:

- Field of view angle
- Ratio of x to y of a window
- Near clipping plane

---

[1]These color values range from 0 to 255. *RGBcolor*(0,0,0) is black and *RGBcolor*(255,255,255) is white.

- Far clipping plane (look distance)
- Position of the viewer's eye
- Point the viewer looks at
- Viewing twist angle about the line-of-sight

The use of these functions is covered in detail later in this chapter when the platform modeling is discussed.

B. TWO-DIMENSIONAL TERRAIN DISPLAYS

After the introduction billboard and the welcome screen[2], a two-dimensional 35 square kilometer map is displayed. This is one of two different two-dimensional map displays. The other is a ten square kilometer map, which is displayed after an area of operation has been selected. The drawing of these two-dimensional maps is quite simple. Starting with the lower left-hand corner of the map and working upward and to the right, a column of points is drawn. The elevation of individual points is used to set the appropriate drawing color, which depends upon the current color scheme selected. The width of the line drawn is such that subsequent lines overlap where the line is too wide. The number of lines drawn is dependent upon the current resolution chosen. The resulting display is a two-dimensional contour map of the terrain. A color scheme key is displayed on the right side of the map.

---

[2]Displayed during the initial reading of the terrain database, which takes approximately 30 seconds to complete. A moving wait bar is displayed during this time. See Chapter III for discussion of database access.

C. OVERLAYS

The overlay display is an enhancement added to MPS II. If any platforms are present on or above the terrain, their location is displayed with an overlay on both the ten and 35 kilometer two-dimensional maps. The original MPS had no overlay for the 35 kilometer map, and the ten kilometer display of platform positions was not drawn in the overlay mode. It was important to convert MPS II to draw these in the overlay mode because the two-dimensional maps take on the order of ten to 30 seconds to draw. Obviously, it is not possible to redraw the map each time through the loop. Consequently, after the map is drawn once, the function called *drawmode*(OVERDRAW) is made to begin drawing in the overlay mode. The IRIS 4D/70GT has a two bit overlay buffer. When in overdraw mode, the overlay buffer is drawn into instead of the back display buffer. Overlay drawings are usually simple and fast, but, because there is only one overlay buffer, we cannot use double buffering for drawing in the overlay planes. Since the overlay is repeatedly drawn, erased, and redrawn, the overlays do experience some flicker. This flicker is not all bad since it helps bring attention to the motion of the relatively small icons.

Another limitation with the overlay mode is the limit of four colors. MPS II uses the convention of red icons for ground platforms, black icons for air platforms, and blue for direction arrows and observation posts. Clear is the last color, and it is used to erase the overlay plane.

1. Thirty-five Kilometer Map

Adding the overlays to the 35 kilometer map was important for several reasons. The first is because platforms are no longer limited to a ten kilometer area and can drive out of the area of operation. The overlay is important to be able to pick the proper ten kilometer area the platform moved into. Second, since the simulator is used to display real-time platforms, the overlay is important to be able to initially select a ten kilometer area of operation where vehicles are located. Third, since the platforms are moving in real time, the overlay helps the operator get the big picture of where all platforms are located and where they are going. Platforms are displayed as small squares (which appear as dots) in the colors mentioned above.

2. Ten Kilometer Map

The original MPS displayed platform icons on the two-dimensional ten kilometer map. This map was not drawn in the overlay mode. This allowed MPS to use more colors since the number of colors is not limited in the NORMALDRAW mode.

In MPS II, the platforms are displayed as icons that are shaped like the type of platform each represents. In addition, an arrow pointing in the platform's current direction of movement is displayed.

D. THREE-DIMENSIONAL TERRAIN DISPLAYS

1. Background

The original MPS contained a new algorithm for displaying the three-dimensional terrain. Only the terrain in the field of view is drawn, and the terrain is drawn from the viewer's eye all the way to the limit of the ten kilometer area of operation. A distance attenuation algorithm was implemented which takes advantage of the fact that one can see less detail at greater distances. This resulted in significant performance increases [Ref. 1:pp. 36-49].

The basic strategy of the drawing algorithm used by MPS was to use a data structure designed for speed, and to precompute as many static values as possible. Then MPS takes advantage of the CPU's computational power, and computes "where to draw" each time through the drawing loop to reduce the amount of terrain drawn to the greatest extent possible. This method turns out to be effective. During the development of MPS II, tests were done to see whether greatly simplifying the "where to draw" computations at the expense of drawing more terrain would result in an increase in speed. In fact, the opposite occurred, and the result was a slower drawing algorithm.

2. MPS Terrain Data Structure

As mentioned above, the terrain data structure was designed for speed. The terrain is drawn as a series of squares formed by drawing two triangles. MPS stored the terrain as triangles, storing the x, y, and z coordinates of every vertex of every

triangle. This was done to expedite the drawing routine. See Figure 5-1 to see the construction of these triangles.

This data structure worked very well for MPS, but sacrificed memory space for increase in drawing speed. Since the IRIS 4D/70GT at the Naval Postgraduate School has eight megabytes of RAM, this was never a problem. But, since MPS II is designed to display high-resolution terrain, this became a very big problem. Examining the terrain data structure more closely, it can be seen that by storing each vertex of each triangle making up the terrain, the vast majority of terrain data points are stored six times. This is because each interior[3] data point is the vertex of six different triangles (see Figure 5-2). This was not affordable with higher resolutions.

3. New Terrain Data Structure

The goal for MPS II was to simplify the terrain data structure, but not sacrifice speed. Although the IRIS 4D/70GT has a virtual memory management system, if the terrain data structure could not fit in the internal RAM, and disk swapping occurred during the display loop, no significant speed would ever be accomplished. The first obvious modification to the data structure was to only store each terrain data point once. This decreases the size of the data structure by a factor of six. See Figure 5-3 for detail on the new data structure.

---

[3]An interior data point is a point not on the outside edge of the database. Points on the edge are stored three times each except the corners which are stored either twice or once, depending on the corner.

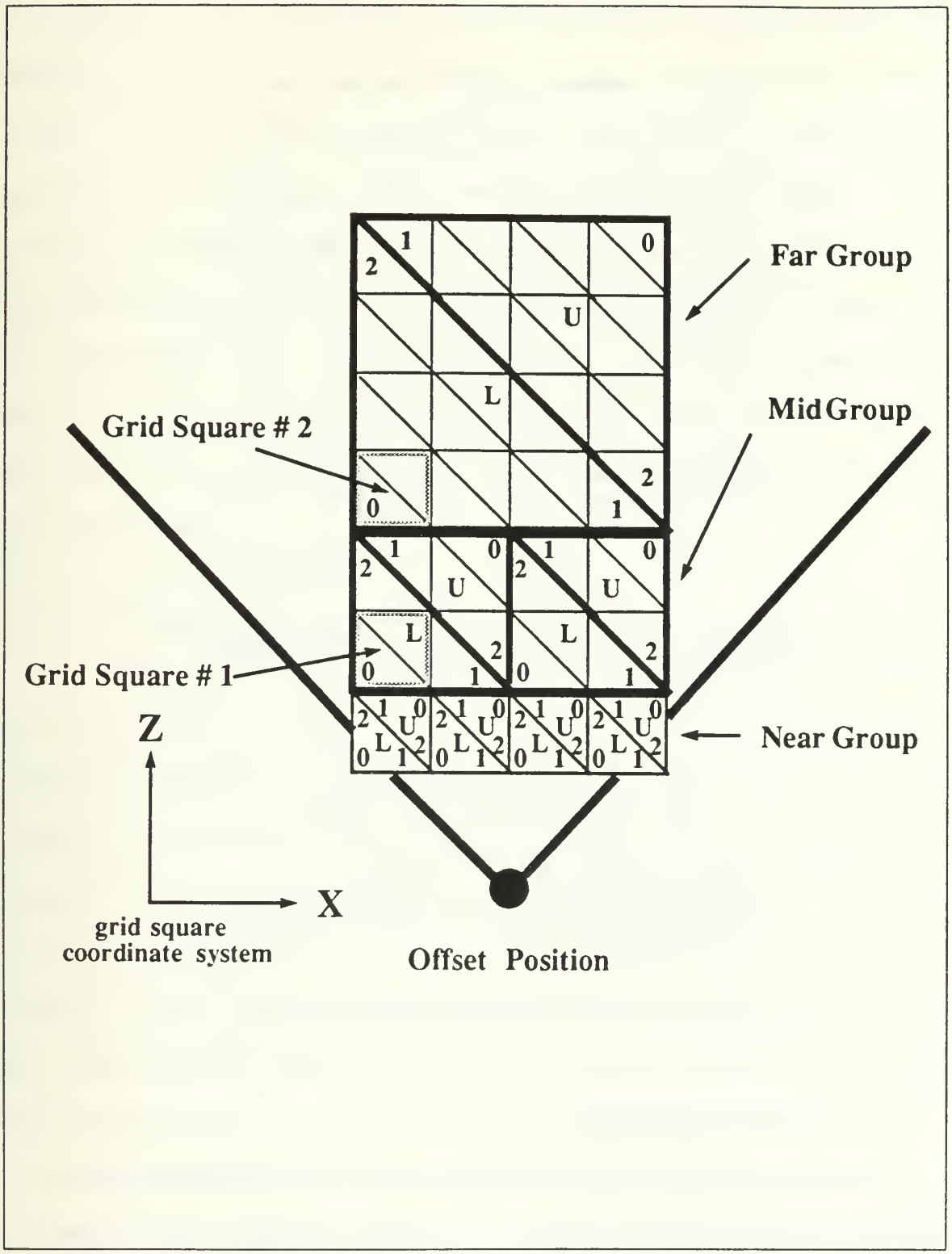


Figure 5-1 MPS Terrain Construction

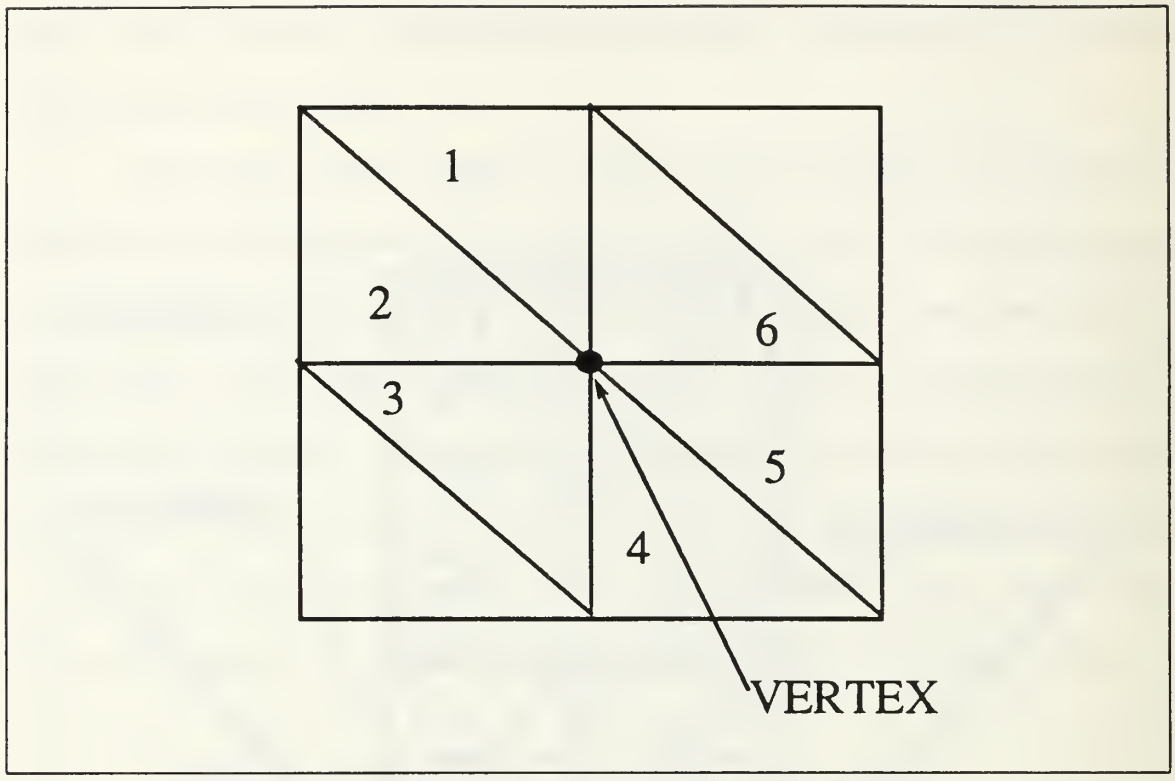


Figure 5-2 One Point is Vertex For Six Triangles

gridcoord2[Z][X][GRIDCOORDS]

where: Z is the vertical index into the array
X is the horizontal index into the array
GRIDCOORDS are the x, y, and z coordinates of the data point

Figure 5-3 MPS II Terrain Data Structure

4. Mesh Drawing Primitive

Although considerable time was spent trying to increase drawing speed by simplifying the calculations of the drawing boundaries, this effort proved fruitless. Instead, improved methods of drawing were examined. Included with the other system

drawing primitives for drawing points, lines, and polygons is a primitive routine for drawing a *mesh* [Ref. 9]. A *mesh* is defined as a series of triangles with common vertices. To use the *mesh* primitive, a series of vertices is sent to the drawing primitive. After three vertices are sent, the mesh function draws a triangle. As it receives each successive vertex, the last three vertices are connected to form a triangle. By using the same algorithm used by MPS for calculating what terrain to draw, and then implementing routines using the mesh primitive, an increase of three to five frames per second was realized. This routine also made effective use of the smaller and simpler data structure explained above. See Figure 5-4 for details of how the mesh primitive is used.

5. Distance Attenuation

The distance attenuation algorithm used by MPS is also used by MPS II [Ref. 1:pp.111-124]. Simply put, terrain at greater distances is drawn at lower resolutions; therefore, fewer polygons are drawn and the display loop is faster. To accomplish this, the viewing sector is divided into three regions, near, mid, and far (see Figure 5-5). In the near region, the terrain is displayed at full resolution. In the mid region, four adjacent squares are combined to form one square. In the far region, four of the mid region squares are combined into one (i.e., 16 near region squares). The size of these regions is dependent upon the zoom angle. The smaller the zoom angle (higher magnification), the further full resolution terrain is drawn.

At the boundaries of these groups, small gaps can be formed where the polygons no longer meet edge to edge. As was done in MPS, these gaps are filled in by drawing a triangle formed by the three vertices forming the gap, using the normal

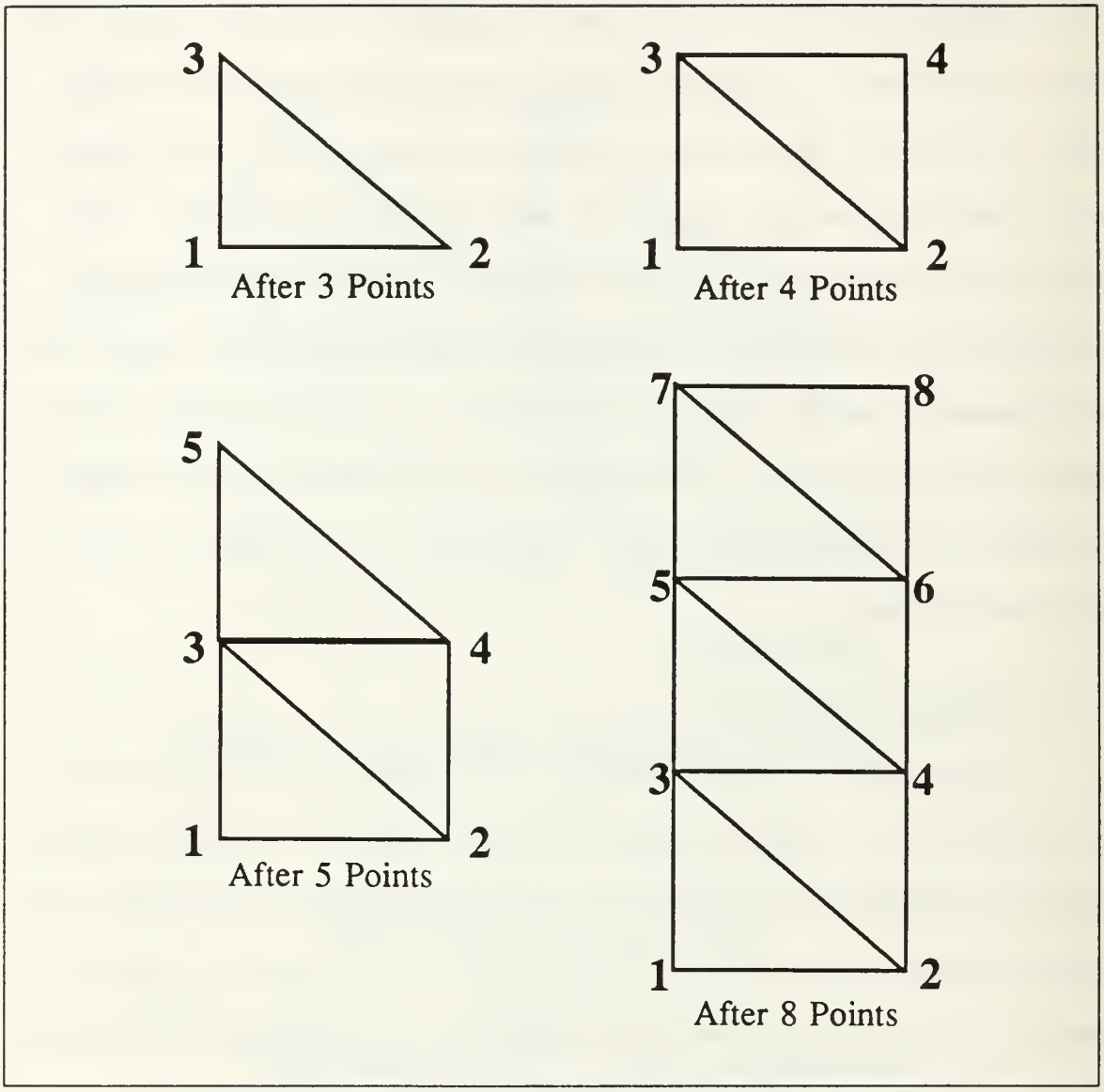


Figure 5-4 Drawing With the Mesh Primitive

vector and material definition of an adjacent polygon. The option for displaying the holes in a different color was not implemented in MPS II.

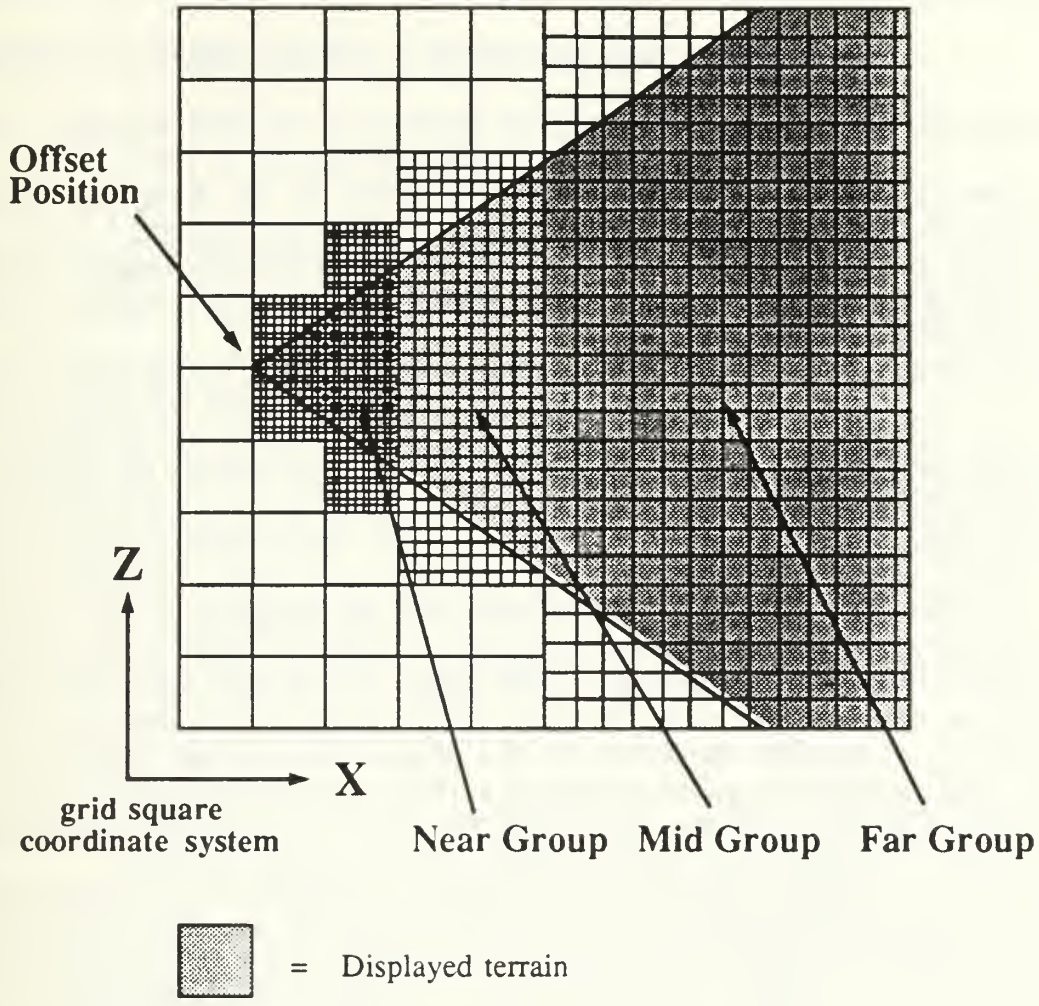


Figure 5-5 Distance Attenuation Scheme

6. Terrain Normals

a. Background

The authors of MPS implemented a real-time lighting and Gouraud shading model using the specialized graphics hardware on the IRIS 4D/70GT. They implemented a simple model of the sun moving across the sky during the different times of day and months of the year. This provided for realistically lighted platforms and terrain. [Ref. 1:pp. 28-36]

The displayed color of each polygon of the terrain and platforms is a function of its defined material characteristics and the angle between the direction of the light source and the normal vector of that polygon. The material definition of the terrain in MPS II is the same as that of MPS with the exception of how the RGB components of that color are assigned[4]. MPS assigns one of eight major color values to each range of possible elevations. It then assigns one of these colors to even-numbered grid squares, and one of eight minor colors to the odd-numbered squares. This results in a checkerboard effect on the terrain. Since there are no cultural terrain features available as visual reference points, the checkerboarding is useful when a platform is traversing flat terrain to give the operator a sense of motion. Unfortunately, this effect also detracts from the realism of the terrain.

Normals for all the triangles making up the terrain are precomputed, and then used by the special purpose hardware to determine the final display color of the polygon. The RGB component values of any one square (two triangles) are

---

[4]The material definitions of the platforms in MPS II are exactly the same as in MPS.

determined by the elevation of that square's lower left-hand elevation data point. The actual color of the two triangles of the square might be different depending upon the normals of that triangle. But the color of any one triangle is always constant throughout the triangle.

b. MPS II Terrain Normals

Research into the display of more realistic terrain lead to the study of different methods of calculating the normal vectors used by the graphics hardware in computing realistically lighted and shaded terrain. The first key was to use **vertex normals** instead of polygon normals. A vertex normal is a vector pointing perpendicular to the plane of the point. But, what does it mean to take the normal of a point? Each data point is a vertex for up to six different triangles. First, the normals for all of these triangles are computed and then normalized to unit vector length. Then these vectors are averaged to form the **true vertex normal** of the data point. This normal takes into account the normals from all surrounding polygons.

As can be seen, this operation can be computationally intensive, and although these values are all computed prior to the three-dimensional display loop, ways of simplifying the computations were examined. Initially, for the sake of simplicity, the normal of just one triangle was computed and used as the vertex normal. Then, other approximations of true vertex normals were implemented, as well as a routine to calculate true vertex normals. These methods were compared for both their speed of computation and for the realism they added to the display. The gross approximation of the vertex normals provided a significantly better picture than polygon normals. Also, there was little difference between the terrain display using the

different approximations of vertex normals and that of the terrain drawn with true vertex normals. With low resolution terrain data, the time differences between the methods was not significant, but became so as resolution was increased. See Figure 5-6 for further details on the calculation of normals.

c. MPS II Vertex Normal Data Structure

In addition to providing for more realistic-looking terrain, the use of vertex normals also simplified the data structure used to store the normals. Instead of the complex system used by MPS, a simple two-dimensional array was used to store the normals. This also made indexing the normals a matter of using the same index as used for the data point itself.

7. The MPS Three-dimensional Terrain Display Algorithm

MPS II implements much of the original MPS drawing algorithm. This section explains the portions of that algorithm that are unchanged in MPS II.

The first step is to determine the area within the field of view (see Figure 5-7). This field of view is expanded to ensure enough terrain is drawn by offsetting the viewer's position. The offset distance is a function of the field of view; the smaller the field of view, the larger the offset.

The next step is to determine the drawing direction. The distance attenuation algorithm is affected by the viewing direction. For this reason, the terrain is drawn in one of four different directions, depending upon the viewing direction (see

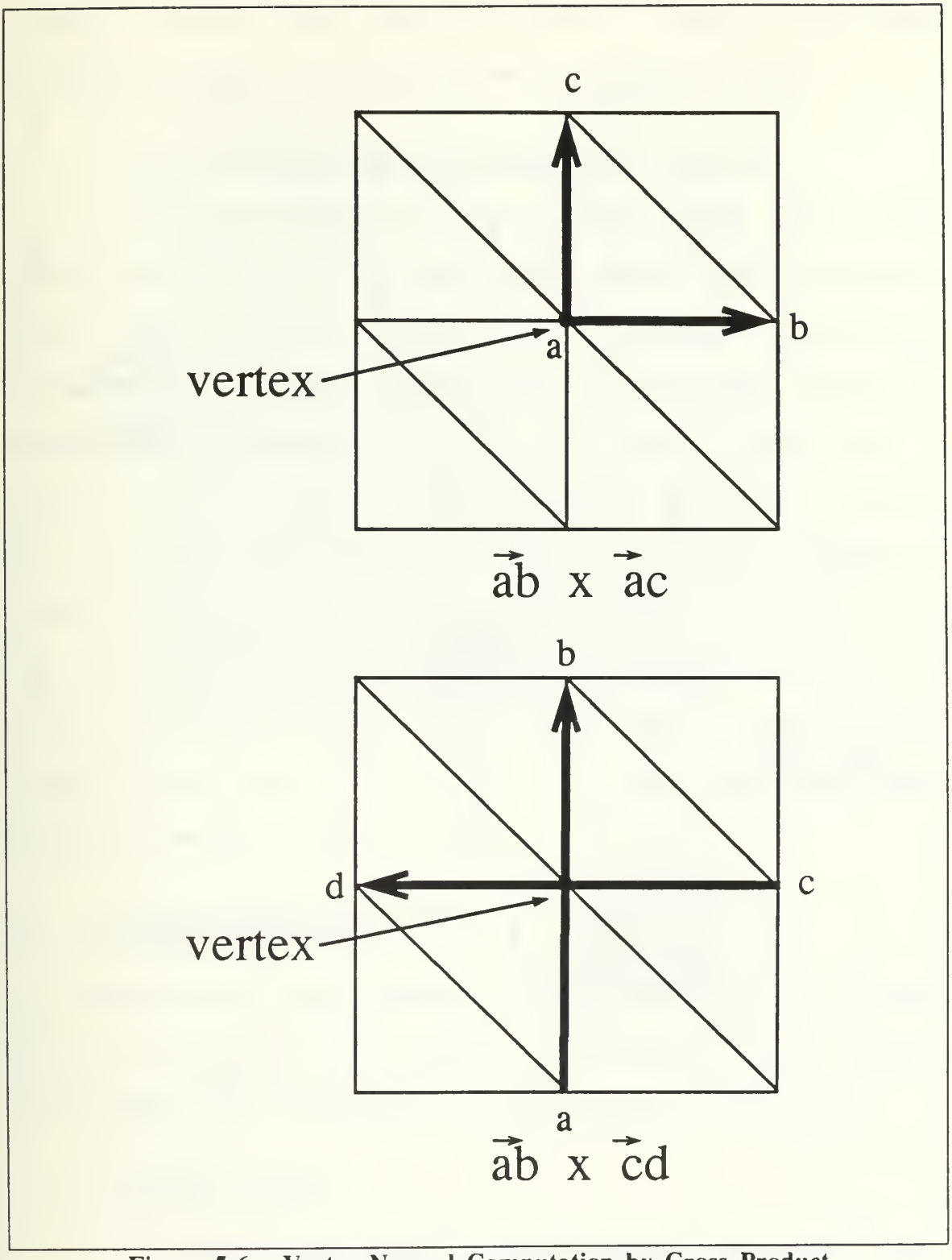


Figure 5-6 Vertex Normal Computation by Cross Product

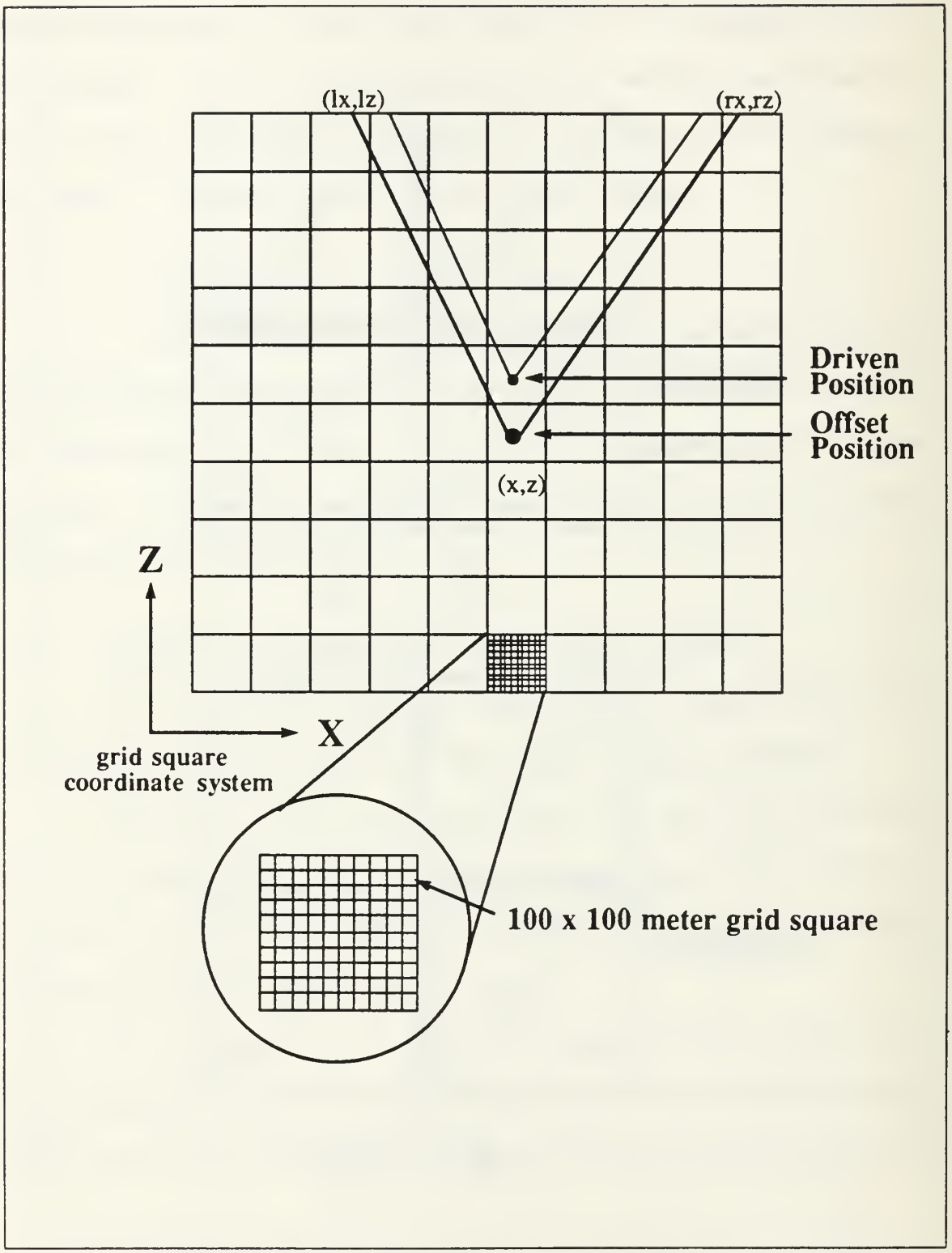


Figure 5-7 Determining Field of View

Figure 5-8). Next, the limits of the drawing groups are computed. The final step is to bind the individual vertices with the proper material definition (color).

8. MPS II Three-Dimensional Terrain Display Algorithm

Using the *mesh* primitive and *vertex normals*, plus the "where to draw" routines from the original MPS, the display algorithm used by MPS II becomes quite simple. Figure 5-9 provides an overview of the steps required to display the three-dimensional terrain. Simply stated, the "where to draw" routines return the row numbers and the squares within those rows. A pair of vertices is passed to the *mesh* routine at a time, even though the mesh primitive groups three vertices. The key is that it uses the **last three vertices** it has received. After the first two vertices are passed to the mesh, no triangles are drawn. But, on during the next iteration of the loop, two more vertices are passed to the mesh, and a triangle is drawn after each vertex. Each vertex is bound to a material color, and the vertex normal is passed along with the coordinates of the vertex itself. The code that accomplishes the three-dimensional drawing is in file *drawterrain.c*.

E. PLATFORM MODELING

The model used to display platforms in MPS II is similar to that in the original MPS. Each time through the draw loop (either two or three-dimensional) the position of each platform must be updated from its position during the previous frame.

1. Platform Position

A platform's new position is calculated by multiplying the platform's speed by the elapsed time since the last frame to get a distance traveled. The platform's

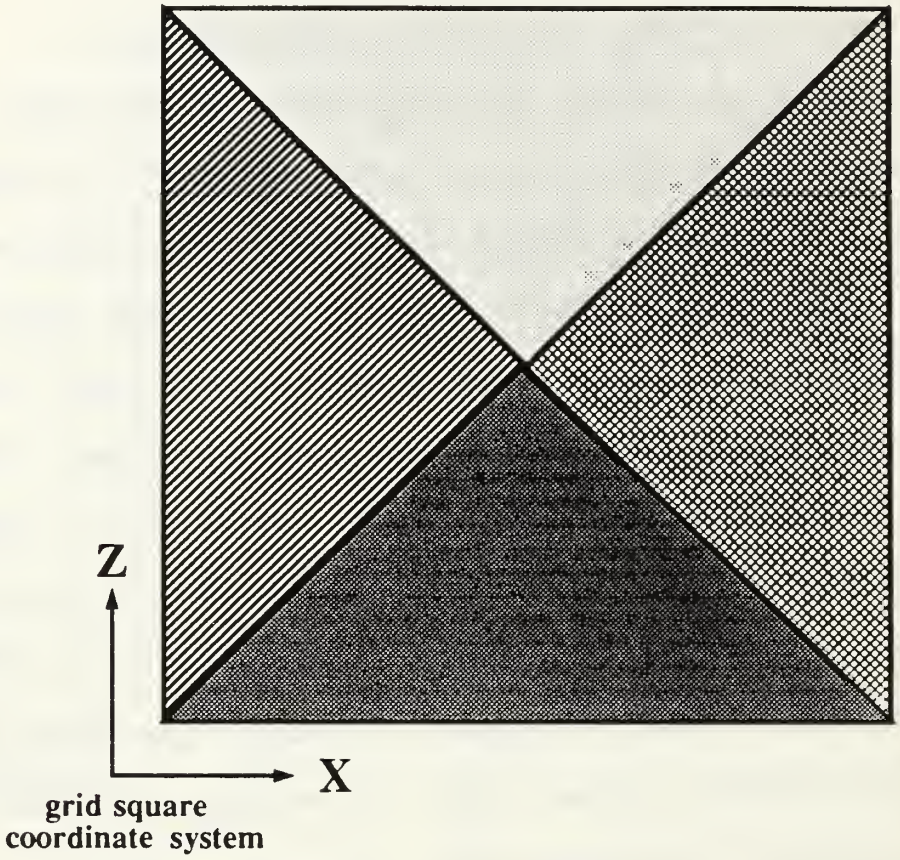

Z
X
grid square
coordinate system
= Look direction North. Draw min to max z.
= Look direction South. Draw max to min z.
= Look direction East. Draw min to max x.
= Look direction West. Draw max to min x.

Figure 5-8 Determining Viewing Direction

```
for (j = FIRST_ROW; j < LAST_ROW; j++)
{
        Begin Mesh
                for (i = BEGINNING_OF_ROW; i < END_OF_ROW; i++)
                {
                        Bind Material
                        Send Vertex Normal Point 1
                        Send Vertex Coordinates Point 1

                        Bind Material
                        Send Vertex Normal Point 2
                        Send Vertex Coordinates Point 2
                }
        End Mesh
}
```

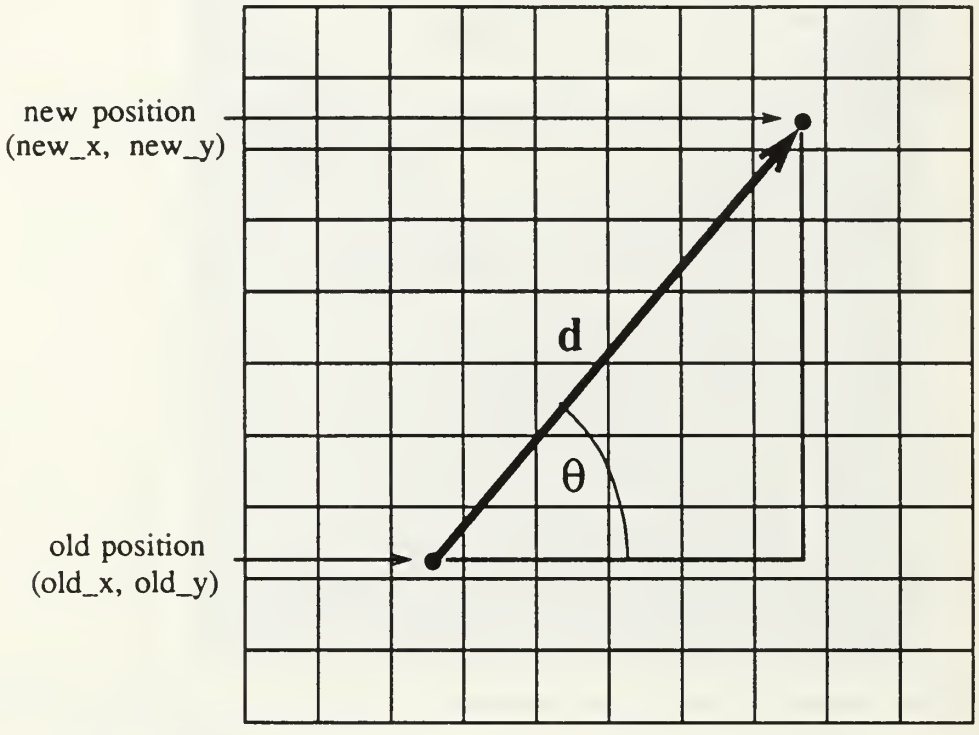
Figure 5-9 Mesh Drawing Routine

position is then updated by moving the platform in the direction of travel the calculated distance traveled. See Figure 5-10 for the formulas used in this calculation.

2. Platform Orientation

For land vehicles, the orientation of the vehicle is a function of the terrain it is traversing. With the vehicle axis established as shown in Figure 3-1, the following conventions are established for vehicle rotation:

- **azimuth** - Rotation about the Y-axis in right-hand sense from positive X-axis. *Counterclockwise* as you look down the positive Y-axis toward the origin. Also called the platform's course.
- **pitch** - Rotation about Z-axis in right-hand sense from positive X-axis. *Counterclockwise* as you look down the positive Z-axis.
- **roll** - Rotation about X-axis opposite to right-hand sense from positive Z-axis. *Clockwise* as you look down the positive X-axis toward the origin.
- **heading** - Compass courses. *Clockwise* angle in degrees between north and the platform's X-axis. Not used except for the user during display.



Distance Traveled = velocity x elapsed time

$$new\_x = old\_x + (d \times \cos(\theta))$$

$$new\_y = old\_y + (d \times \sin(\theta))$$

Figure 5-10 Updating Platform's Position

The data structure used for the platforms includes fields for both *pitch* and *roll*. These values are calculated as follows. First, the elevation of the terrain of the platform's UTM coordinates is computed. Next, the elevation of the terrain 20 meters directly in front of the vehicle is computed. The slope of the terrain in the direction of travel is then computed using these two elevations. This value is used for the *pitch* of the platform. The value for the platform's *roll* is computed in a similar fashion using a point 20 meters to the side of the vehicle. No pitch and roll values are computed for airborne vehicles (FOGM missiles) in MPS II.

The original MPS used an offset of 1.5 meters from each platform to compute the pitch and roll. Because the terrain is represented as a series of planes, the intersections of two polygons can be very abrupt, and the smooth tops of hills are lost. For this reason, as the vehicle crosses the boundaries of polygons at the top of a hill, pitch and roll values change from positive to negative immediately. This results in jerky motion. By extending the offset used to calculate the pitch and roll, this effect was dampened as the effect of the next polygon is taken into account earlier so the pitch and roll begins to change from positive to negative before the edge of the polygon is actually reached by the platform. The result is a much smoother motion, but over very rough terrain, the tops of hills are clipped by the platform's position changing early. This is only noticeable looking at the platform from another platform.

With both the location and orientation of each platform established, the vehicle can be properly displayed. MPS used a structure of a linked list of platforms located in each grid square to better keep track of which vehicles to draw. MPS II

simplifies matters and simply draws all platforms located in the ten square kilometer area of operation. This method is much simpler and is just as fast as the old method.

During the update computations described above, the elevation of the terrain at a particular point is computed by a routine in file *gnd_level.c*. This routine does a linear interpolation from the elevation data points to the location in between them. For this reason, care must be taken not to attempt to calculate the elevation, pitch, or roll of platforms outside the ten kilometer area of operation, because no terrain elevation data is in the system memory. To ensure this does not happen during the update procedure, the UTM coordinates of the platform are first updated. Then a check is done to see if the platform is in the ten kilometer area. If it is not, a flag is set in the platform's data record, and the elevation, pitch, and roll are not calculated.

3. Viewing Perspective

a. Background

The viewing perspective in the original MPS used a simple model that does not give a realistic-looking view with respect to the motion of the platform for ground vehicles. The three-dimensional view that is seen is determined by the call to the two functions *perspective*() and *lookat*(). How the parameters to these functions are computed is the key. In the original MPS, the viewer's lookat position stayed relatively constant, regardless of the orientation of the vehicle. This was fine for vehicles traversing relatively flat terrain, but when traversing steep terrain, it gave a totally unrealistic view. For example, when traveling up a steep incline, the vehicle would have the correct pitch, but the lookat point would be such that all that could be

seen was the hood of the vehicle. In other words, the lookat point did not change with respect to the orientation of the vehicle as is the case in real life[5]. Another problem was that the viewer's eye position was always a fixed distance, vertically above the point on the terrain where the vehicle was located. We can see by Figure 5-11 that this should not be the case, and that using this method, the eye position could actually end up outside the vehicle.

b. Coordinate System Transformations

The solution to these problems was two-fold. First, the eye position had to be kept in a fixed location with respect to the jeep (see Figure 5-12). Secondly, the lookat position must be adjusted with respect to the gradient of the terrain on which the vehicle was positioned.

Since the eye position is fixed with respect to the platform, the eye position is known, but this position is in terms of the platform's or *body* coordinate system which is translated and rotated. The eye position must be transformed to the system's *world* coordinate system. These transformations could be manually computed [Ref. 10], but fortunately, the system hardware is custom designed to do just this type of transformation (although it is not normally used to transform the eye position). The function *transform_body_to_world*() in the file *math_utilities.c* performs these transformations. See the actual code in Figure 5-13.

---

[5]The viewpoint of a passenger in a moving platform obviously does not stay fixed with respect to the orientation of the platform. But in general, as a platform pitches and rolls, so does the passenger and his corresponding view. So this is a relatively good model when combined with the other viewing controls available, i.e., view direction and view tilt.

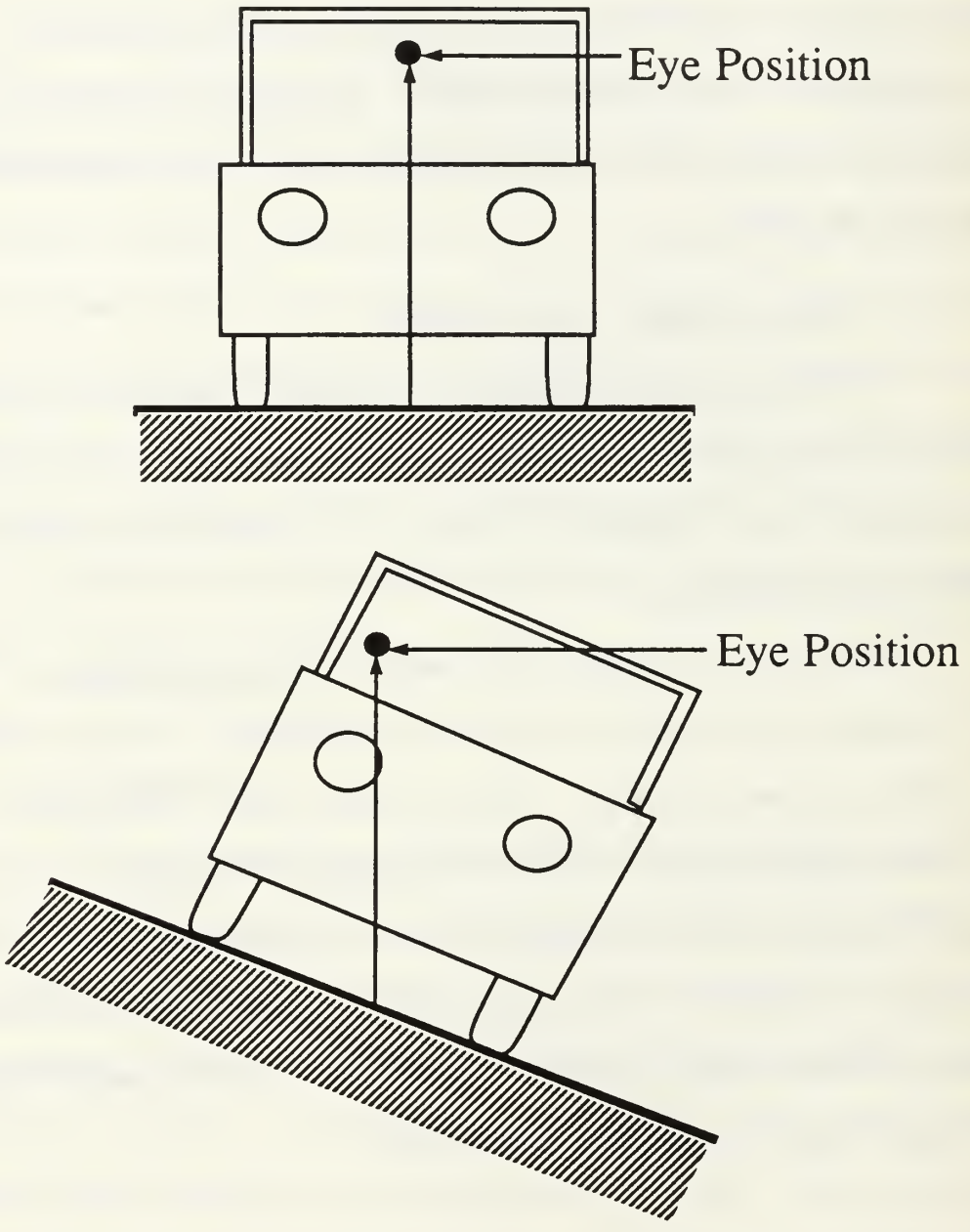


Figure 5-11 Eye Position of MPS

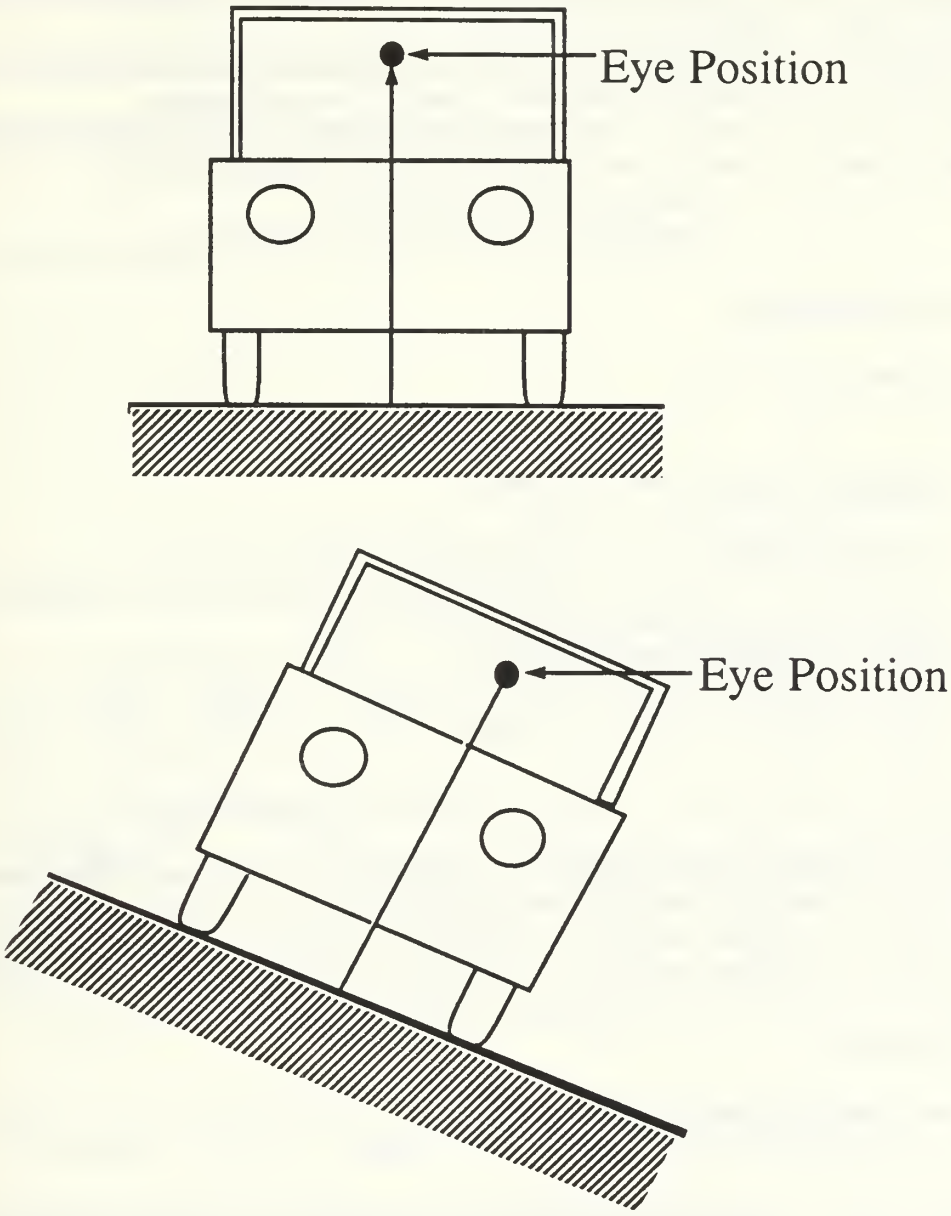


Figure 5-12 Corrected Eye Position of MPS II

```
/*****************************************************************
  FUNCTION: transform_body_to_world
  PURPOSE    : Transforms coordinate in body axis to world coordinates.
  Uses Iris matrix multiply microcode to avoid sins and cosines.
*****************************************************************/
transform_body_to_world( float azimuth, float elevation, float roll,
                         float dx, float dy, float dz,
                         float *eye_x, float *eye_y, float *eye_z )
{
    Matrix offset_mx;

    pushmatrix();


    loadmatrix(unit);  /* Load unit matrix */

     /* Assumes platform's nose points along positive X axis */
     /* P(world) =  P'(body) * ROT(azimuth) * ROT(elevation) * ROT(roll) */
     /* Do rotations in reverse gimbal order */
     rotate( (Angle)(azimuth*RTOD*10.0),       'Y' );  /* azimuth */
     rotate( (Angle)(elevation * RTOD * 10.0),'Z' );  /* roll     */
     rotate( (Angle)(-roll * RTOD * 10.0),     'X' );  /* elev     */

     getmatrix( offset_mx );  /* Get accumulated rotation matrix */

     /*  Pre-multiply rotation matrix by offset vector to get world coordinates. */
     *eye_x = dx*offset_mx[0][0] + dy*offset_mx[1][0] + dz*offset_mx[2][0];
     *eye_y = dx*offset_mx[0][1] + dy*offset_mx[1][1] + dz*offset_mx[2][1];
     *eye_z = dx*offset_mx[0][2] + dy*offset_mx[1][2] + dz*offset_mx[2][2];

     popmatrix();

}  /* end transform_body_to_world  */
```

Figure 5-13 Transforming Platform Coordinates to World Coordinates

Adjusting the lookat position to give a more realistic view is a similar problem. The lookat point is simply a target at a constant distance along the X-axis

of the viewer. By performing the same homogeneous transformation on this point, which is in *body* coordinates, the lookat point coordinates, in system *world* coordinates, is returned. This corrects simplifications in MPS that neglect the viewer's pitch and roll in determining the point of view (see Figure 5-14).

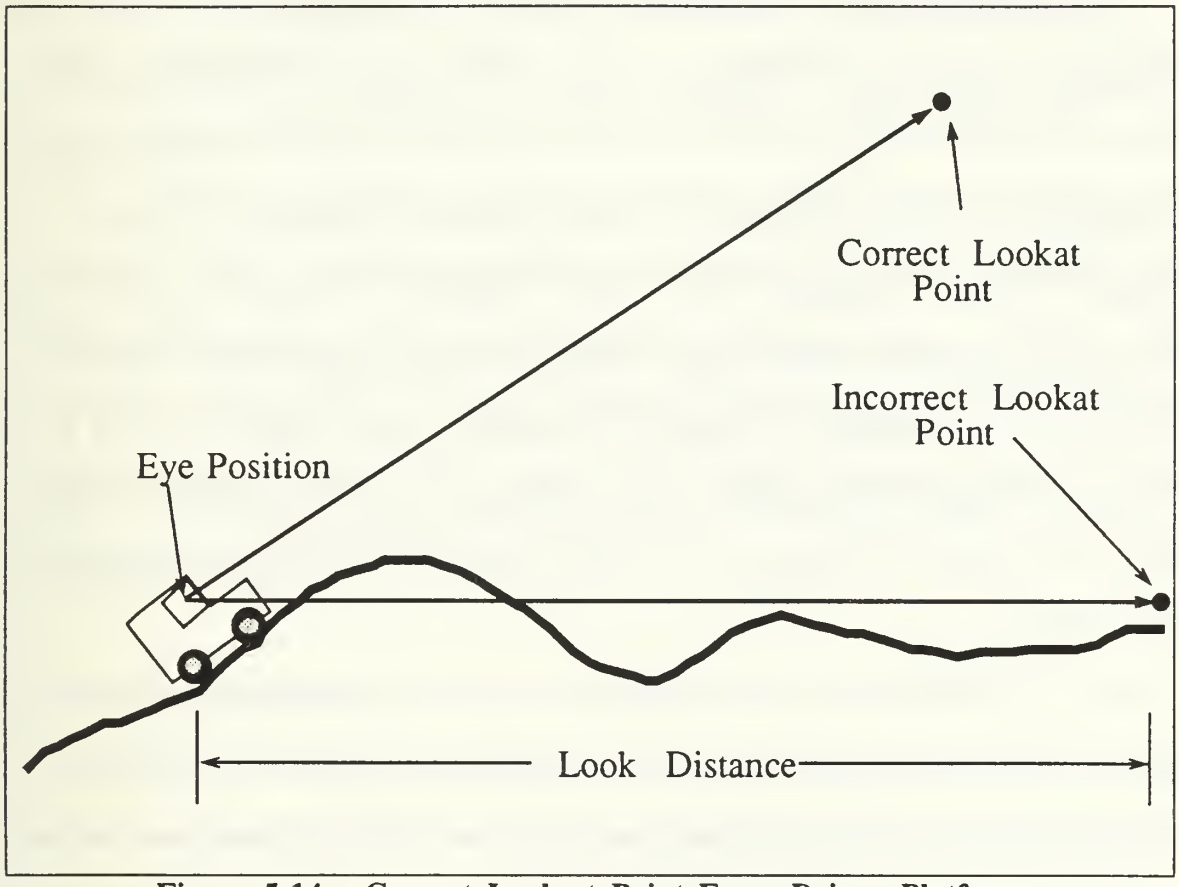


Figure 5-14 Correct Look-at Point From Driven Platform

VI. NETWORKING CAPABILITIES OF MPS II

A. BACKGROUND

The networking code of the original MPS has been overhauled in MPS II. MPS used *blocking network I/O* and *shared memory* to handle some of its networking tasks. This means that when MPS checked the network interface for incoming messages, and no messages were received, MPS waited (blocked) until a message was received. To keep this blocking mode from slowing down the simulator, MPS spawned a separate process (*network_receive*) to listen to the network and pass the incoming information to MPS. The two processes (MPS and network_receive) communicated by using an area in the system's memory called *shared memory*. The network_receive process would write incoming information into the shared memory, and MPS would read the shared memory.

Users of MPS had to be very careful when operating in the networking mode. Since use of the shared memory was not coordinated between the two modules, an incoming message placed in the shared memory space could be overwritten by a subsequent incoming message before the first message was read by MPS. This was especially true if many platforms were employed in the simulation, and their velocities often changed, generating network message traffic. Additionally, all simulators on the network needed to be in the three-dimensional mode before the next simulator was started, and the subsequent simulators were limited to using the same 10 kilometer area of operation the first simulator was using. Finally, any time one simulator went into

a pop-up menu, that simulator's platforms were stopped on all other simulators on the network.

To simplify the networking procedure, MPS II uses *non-blocking network I/O*. By using non-blocking I/O, a spawned process is not needed, and neither is the use of shared memory. When checking the network interface, a process using non-blocking I/O continues processing even when no messages have been received from the network. As a side benefit of this change, the program execution is much easier to understand and to debug.

None of the restrictions mentioned above for MPS apply to MPS II. Simulators can be in any mode of operation when a new simulator enters the network environment. The implementation of UTM coordinates allows different simulators to be in any 10 kilometer area of operation; all platforms are displayed in their proper location. The real-time model eliminates the need to inform other systems when pop-up menus are entered.

B. ARCHITECTURE

Many of the networking concepts from MPS are still implemented in MPS II, but most were modified to be more efficient or to make the code easier to understand. Like MPS, MPS II establishes a connectionless link among simulators using the *User Datagram Protocol/Internet Protocol (UDP/IP)* protocols, and uses a predefined set of messages to communicate with other running MPS II simulators. Figure 6-1 shows the connections to the LAN and the interprocess communication links among three MPS II simulators.

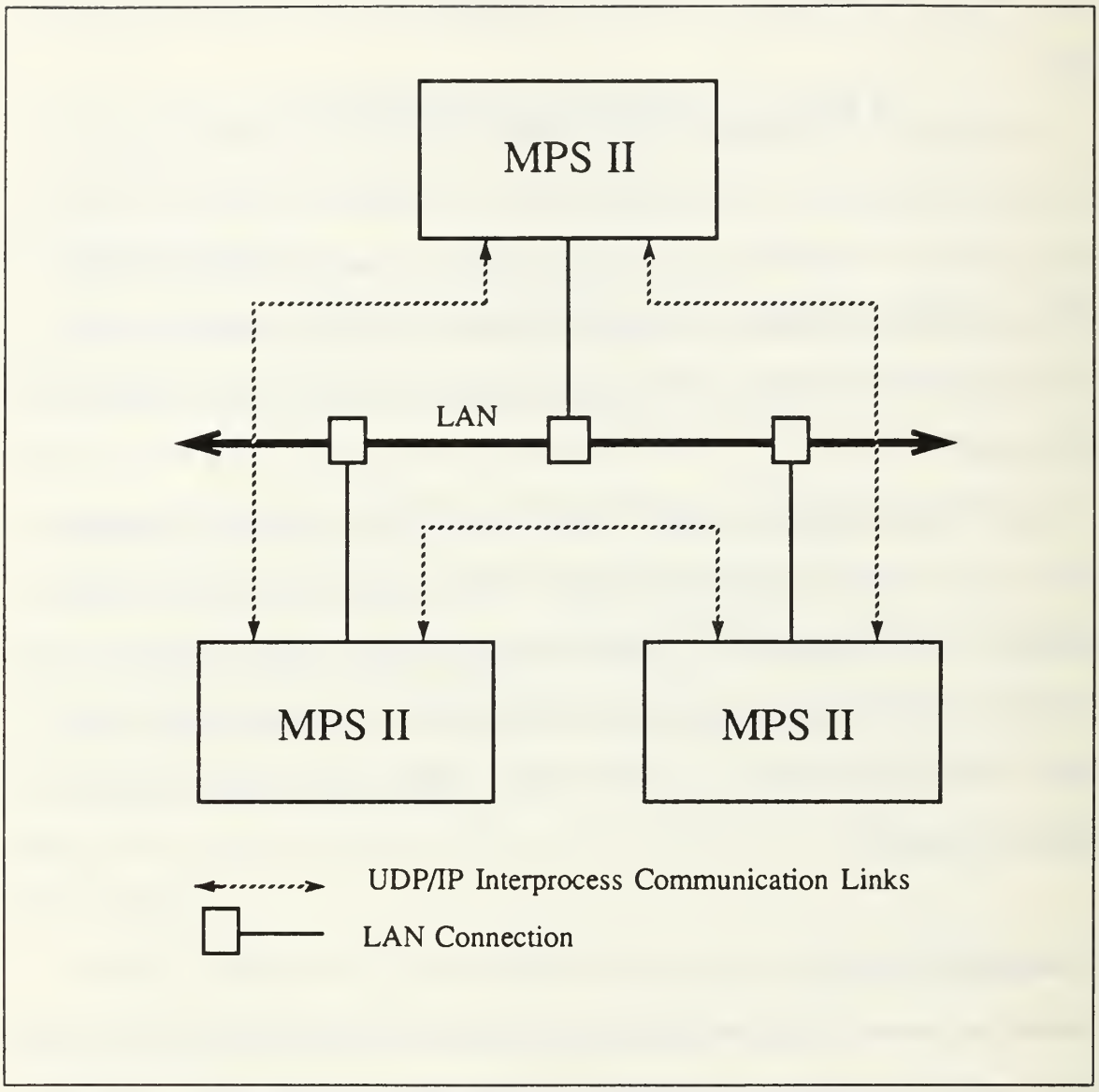


Figure 6-1 Interprocess Communication Links

1. Protocols

The UDP/IP protocols provide an unreliable, connectionless delivery service. To transmit a message, the protocols divide the data into packets. The error-free, ordered delivery of these packets is not guaranteed [Ref. 14]. Although the unreliable aspect of the protocol is not desirable, the connectionless delivery service is needed. Using a connectionless protocol allows the broadcasting of messages to an unspecified

number of destinations [Ref. 15:p. 41]. In this case, the multiple destinations could be more than one graphics workstation running the simulator, or another machine that further processes the data. The module that determines the intervisibility between two platforms is such a process (see Chapter VIII).

To establish the UDP/IP communication link among processes, a process opens a *SOCK_DGRAM* socket, sets the socket to the broadcast, non-blocking options, and then binds a port number to the socket. To transmit to, or receive from the network, the *sendto* and *recvfrom* system functions are used. When an event occurs that must be reported to other simulators, the appropriate message is immediately created and sent. This task is done by the function *network()*. MPS II only checks the network once each time through the drawing loop for incoming messages. The function *check_for_packets*() accomplishes this task. If incoming messages have been received, the function handles them appropriately. If no messages have been received from the network, MPS II continues the simulation.

2. Data Structures

MPS II uses a predefined set of messages to communicate among the simulators. At a minimum, the message string consists of a 50 character header identifying the message; however, many of the messages include a message body with additional data. If the message contains additional data, the data follows the header in specified format. Although a 50 character header is used, only the first character is needed to uniquely identify the type of the message. The rest of the characters are used to identify the message in English language. Albeit including the English language identification of the message expends network resources, the advantages of

having the human readable text in the message for monitoring and debugging purposes, and the fact that the connection uses less than one percent of the network throughput[6], outweigh the disadvantage. MPS II uses many of the messages used by MPS. However, the format of one of these messages has been altered to meet new requirements and some new messages have been added.

a. MPS Messages

MPS II uses the following messages also provided for in MPS:

- Initialization message
- Answer message
- All message
- MPS End message
- Lock_on message
- Lock_off message
- Destroy message
- Crash message

An **initialization message** is used to inform all running MPS II simulators that another MPS II simulator has just been started, and is going through an initialization routine. As part of this initialization routine, the starting simulator broadcasts the message to all the running MPS II simulators. When a running simulator receives this message, it responds by sending a message back to the starting simulator. This initialization routine provides the mechanism to insure all platforms, in all running MPS II simulators, are uniquely numbered. The initialization message consists of only a header identifying the message.

---

[6]This usage figure was computed by the Computer Science Department's network analyzer during a performance test of the PROCESS_VDB module (see Chapter VII). The module was sending approximately the average amount of data transmitted during an actual exercise at FHL.

An **answer message** is the message a running simulator responds with when it receives an initialization message. In addition to the message header, the message contains the local simulator's base identification number. This number, unique to each running simulator, is a multiple of 10,000, and is used by the simulator to uniquely number its platforms. The starting simulator adds 10,000 to the maximum base identification number of all the running simulators to determine its unique base identification number. Unfortunately, this scheme does not guarantee the unique numbering of the platforms. Since the messages are sent using the UDP/IP protocols, their delivery is not guaranteed. Also, if any of the running simulators do not immediately respond to the initialization message, the starting simulator may complete its initialization routine before it received the running simulator answer message. After completing the initialization routine and assigning the simulator's base identification number, there is no mechanism to change this number if an additional answer message is received. In an effort to ensure this failure does not occur, the starting simulator sends out five initialization messages, waiting one second and checking for answers after sending each one.

For details on the **All message, MPS End message, Lock_on message, Lock_off message, Destroy message,** and **Crash message,** see [Ref. 1].

b. New Messages

MPS II also uses some messages not provided for in MPS. Although the majority of these messages are used to control the NETWORK_SIMULATOR and PROCESS_VDB modules (see Chapter VII) and to communicate intervisibility data (see Chapter VIII), one, the update message, originated in MPS but was modified to

meet the needs of MPS II. The messages not common to MPS but used in MPS II are:

- Update message
- PROCESS_VDB kill message
- PROCESS_VDB end message
- PROCESS_LOS kill message
- PROCESS_LOS end message
- Observer LOS message
- Target platform intervisibility message

An **update message** is sent by the MPS II module to inform all other processes about a platform's position and velocity at an instance in time. Although normally sent when the platform's velocity changes, it can also be sent when a specified period of time has elapsed during which the platform's velocity has not changed. Since MPS II moves the position of a vehicle among updates using a *dead reckoning* scheme where the vehicle movement is calculated by interpolating from the last known position and velocity, periodic updates assist in maintaining an accurate display. An update message contains the following nine fields:

- Platform's unique identification number
- Type of platform (jeep, truck, tank, etc.,)
- X position of the platform
- Y position of the platform
- Velocity of the platform
- Altitude of the platform
- Direction the platform is traveling
- Time the platform information was recorded
- Incremental count of messages sent for the platform

An example of an update message is given in Figure 6-2, and a description of the message data fields is provided in Table 6-1. When *check_for_packets()* receives an update message, it parses the message to extract its information. It then checks to

"@@ PROCESS VIDS DATA BLOCK MODULE update message @10000.....
.....4.....48766.00000.....69577.00000.....14.705441
.....0.00000.....54.688786.....88000.00000.....23"

Update Message

Figure 6-2 Example Update Message

TABLE 6-1 UPDATE MESSAGE BODY DEFINITION

MESSAGE TYPE	FIELDS CONTENTS	DATA TYPE	FIELD WIDTH
Update Message	Id Number	int	20
	Platform Type	int	20
	X Position	float	20
	Y Position	float	20
	Velocity	float	20
	Altitude	float	20
	Direction	float	20
	Time	float	20
	Message Count	long	20

see if the platform contained in the message is a new platform. If so, the new platform is added to the list of platforms being displayed by the simulator. If the platform is already on the platform list, the platform's information is updated with the message's data. Between receiving messages on platforms, MPS II continues to update the platform's position and orientation using the last information received, i.e., course and speed. Although not needed for display purposes, the incremental count is used to check for lost messages.

The **PROCESS_VDB kill message** and the **PROCESS_VDB end message** are used to control the NETWORK_SIMULATOR and PROCESS_VDB modules (see Chapter VII for further discussion). The **Process_LOS kill message, Process_LOS end message, observer LOS message,** and **target platform intervisibility message** are used to control the PROCESS_LOS module and communicate intervisibility data from the PROCESS_LOS module to MPS II (see Chapter VIII for further discussion).

VII. DISPLAY OF REAL-TIME PLATFORMS

A. SYSTEM ARCHITECTURE

1. Overview

A primary objective of USACDEC was to be able to display, using computer graphics, actual test exercise platforms in real time. MPS II meets the requirements of this objective. However, to provide this capability, the network interface of the original MPS was modified and two totally separate program modules were developed. These modules are the PROCESS_VDB and the NETWORK_SIMULATOR modules. The primary reason the original MPS was modified and the separate modules were developed was to assist MPS II in graphically displaying the platforms in real time. Real time displays are required to create the illusion of motion by the rapid generation and display of still pictures similar to that of motion picture technology. Even with today's advancements in hardware support for graphic displays, this process can be slow, especially when the workstation's processor must also process the data before it can be displayed. By distributing the processing of data among other separate computers and minimizing the amount of processing MPS II must complete before it can display a single still picture, or frame, the display rate is increased. The standard display rate for theatrical motion pictures is 24 frames per second, and at a rate of less than 16 frames per second, the illusion of motion is no longer smooth and appears to flicker and jerk [Ref. 11:p. 3]. In view of the fact that the original MPS displayed,

in the most optimized state, at most 15 frames per second, but sometimes less than one frame per second [Ref. 1:p. 63], a decision was made to remove as much of the non-display processing as possible from the graphics workstation. A second factor that lead to the distributed architecture was insufficient main memory. Depending on the machine, the Silicon Graphics, Inc. IRIS workstations used in this effort contain between eight and sixteen megabytes of main memory [Ref. 12:p. 10]. However, the sum of just the largest data structures used in the MPS II, PROCESS_VDB, and NETWORK_SIMULATOR modules is over 20 megabytes. Although some of the data could be shared by two or all three modules providing an economy if the modules were combined, much of the data is unique to each module. Consolidating the two modules would increase the number of bytes a single processor would have to swap in and out of memory during each display cycle and, again, slow down the display rate. A solution to this problem would be to use a workstation with multiple processors and increased memory. A step in this direction would be to use the Silicon Graphics, Inc., IRIS 4D/240GTX workstation. The 240GTX contains four CPUs, each operating at 25 MHz [Ref. 13]. Table 7-1 lists the largest data structures used in the three modules, their size in bytes, and the modules they are used by. Finally, as a separate module, the PROCESS_VDB module can independently serve more than one MPS II.

2. Purpose of Modules

Although MPS II can function as a lone module or networked with other MPS II modules, it is limited in this mode to displaying internally generated mock

vehicles. Two new modules are needed, in addition to MPS II, to display actual test exercise platforms in real-time. The two new modules work together as a team.

TABLE 7-1 SIZE OF DATA STRUCTURES

STRUCTURE	SIZE	MODULES USED BY
pnterrain	7,699,212 bytes	MPS II
gridcoord	7,699,212 bytes	MPS II
gridcolor	2,566,404 bytes	MPS II
dted	1,283,202 bytes	MPS II
vids_data	1,048,576 bytes	MPS II, PROCESS_VDB, NETWORK_SIMULATOR
in_elev	448,000 bytes	MPS II
player_data	1,520 bytes	PROCESS_VDB
vdb	1,624 bytes	NETWORK_SIMULATOR, PROCESS_VDB

The first of the new program modules is a module capable of accessing instrumentation data available through the FHL Local Area Network (LAN), often referred to as the FHL real-time network, during a test exercise. FHL has a sophisticated array of instruments to record the action during a test exercise. This data is collected by sensors located throughout the test area, relayed back to the cantonment area, and stored and maintained on the computers connected to the LAN. The module reads platform position-location, elapsed exercise time, platform type, and other needed data from the FHL LAN, processes and converts it into a format that MPS II is designed to receive and then transmits the data through the same LAN to MPS II. The instrumentation data is bundled into a complex data structure called the Visual Information Display System (VIDS) Data Block by USACDEC. For this reason, this

module is called the PROCESS_VDB module. Except for two additional interprocess communication messages used to control the PROCESS_VDB module, when running, the module appears to networked MPS II modules as just another networked MPS II module (see Chapter VI for a discussion of networked MPS II modules). This similarity of design allows a networked MPS II module to handle the platform data it receives from the PROCESS_VDB module identically to that received from another networked MPS II module. In fact, the running MPS II module does not even know the data came from a non-MPS II source.

The second module allows the LAN that serves the Computer Science Department at the Naval Postgraduate School to simulate the FHL LAN. This module loads the departmental LAN to appear as the FHL LAN would during an actual test exercise at FHL. This second module was developed as a tool to assist in the development and testing of the PROCESS_VDB module and MPS II's ability to display the test exercise platforms in real time in the school environment. For obvious reasons, this module is called the NETWORK_SIMULATOR module.

3. Interprocess Communication

Essential to the display of the actual test exercise platforms in real time are the interprocess communication links between the NETWORK_SIMULATOR, PROCESS_VDB and MPS II modules. Only with these links can the modules function as a team. Two separate communication links are established between the modules. These links, which are discussed later in more detail, are:

- A connection oriented link between the NETWORK_SIMULATOR and the PROCESS_VDB modules using the *Transmission Control Protocol/Internet Protocol* (TCP/IP).

- A connectionless link between the PROCESS_VDB and the MPS II modules using the *User Datagram Protocol/Internet Protocol* (UDP/IP).

There is no direct communication link between the MPS II and the NETWORK_SIMULATOR modules. Since the only instruction MPS II needs to give the NETWORK_SIMULATOR module is to terminate, and since the PROCESS_VDB module should also terminate when the NETWORK_SIMULATOR terminates, the instruction is sent to the PROCESS_VDB module which then relays it to the NETWORK_SIMULATOR module. Figure 7-1 shows the connections to the LAN and the interprocess communication links between the three modules.

A primary goal of this work was to support USACDEC's operations at FHL, so a conscious effort was made to insure that the software produced is easily ported to FHL and integrated with the rest of their test hardware and software. Central to insuring the work could be easily integrated was the need to use the same network protocols used at FHL. Fortunately, both FHL and the Naval Postgraduate School Computer Science Department use an Ethernet LAN and the Defense Advanced Research Projects Agency (DARPA) Internet protocol suite.

a. Between NETWORK_SIMULATOR AND PROCESS_VDB Modules

(1) Protocols. The purpose of the NETWORK_SIMULATOR module is to provide access, in the school environment, to data that is available through the FHL LAN during a test exercise at FHL. Hence, the module uses the same protocols used at FHL. The data is maintained on a Digital Equipment Corporation VAX 11/780 computer at FHL. Access to the data can only be made by opening a connection to

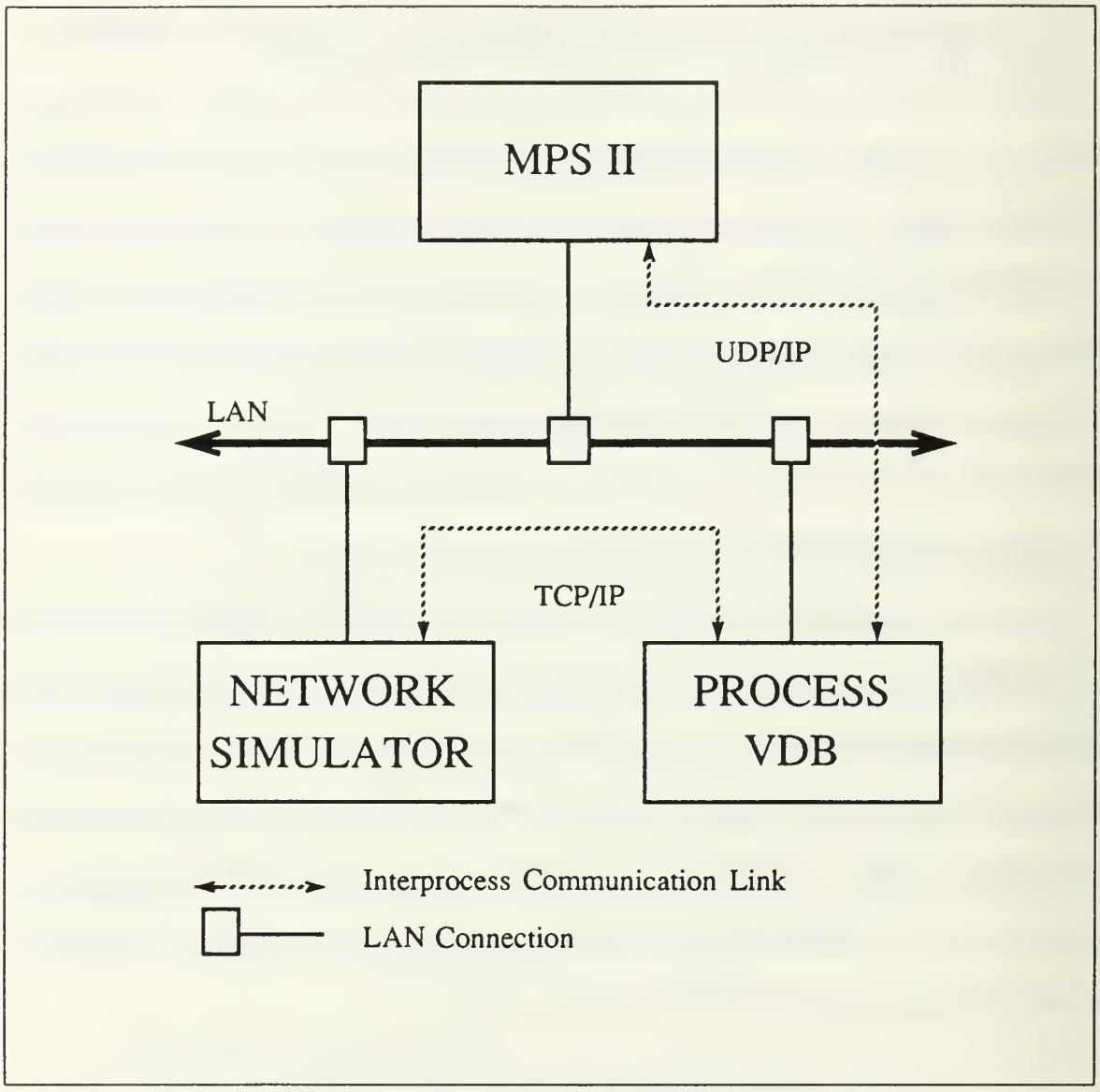


Figure 7-1 Interprocess Communication Links

a process running on the computer using the connection oriented, reliable stream transport service provided by the Internet *Transmission Control Protocol/Internet Protocol* (TCP/IP) protocols. This protocol establishes a virtual circuit between the endpoints and guarantees the error free, ordered delivery of the data. The data is fragmented into small packets for transmission and reassembled at the destination

[Ref. 14:p. 481]. The TCP/IP protocols use a client-server model of interaction to establish this connection.

In this model, the server opens a *SOCK_STREAM* socket in the Internet domain and then waits, listening for a request to establish a connection from the client. The client also opens a *SOCK_STREAM* socket in the Internet domain, but then attempts to connect to the server. When the server receives the request, and it is a proper request, it is accepted and the connection is established. In this case, the NETWORK_SIMULATOR module functions as the server, and the PROCESS_VDB module functions as the client. The connection is full duplex allowing concurrent transfer in both directions. To insure the desired connection is made between the two correct processes, TCP incorporates abstract objects called *ports*. A port uniquely identifies a process on a computer [Ref. 15:p. 137]. Although the prescribed port number at FHL is port number 1025, the NETWORK_SIMULATOR and the PROCESS_VDB use port number 1267. Port 1025 is already assigned to other use at the Naval Postgraduate School, necessitating the difference. These ports are considered unofficially assigned and, unfortunately, UNIX does not provide a mechanism to keep track of unofficially assigned port numbers. However, officially assigned port numbers are maintained in the UNIX system file "/etc/services" and can be obtained using the *getservbyname* system call [Ref. 15:p. 283].

(2) Data Structures. As mentioned above, the instrumentation data is bundled into a complex data structure called a VIDS data block. During an exercise, these blocks are released into the network at a rate that depends on the test exercise scenario. The maximum release rate is one block every 50 milliseconds. Depending

on the type of information contained in the block and the number of platforms active in the exercise, the size of the block also varies. The maximum size of the block is 812 bytes. The block consists of three parts: a block header, a series of predefined *VIDS messages* and a block check word. Figure 7-2 shows the structure of one VIDS data block. Each VIDS data block is separated by at least one word of filler.

The header of the VIDS data block has a fixed structure that encompasses the first 24 bytes of each VIDS data block. The header consists of the following eight fields:

- SYNCHRONIZATION WORD: The eight characters "VIDS0000" that denote the start of a VIDS data block. It is used to identify the start of a VIDS data block.
- INTERNAL SEQUENCE NUMBER: A four byte sequence number field that is no longer used.
- FILLER: A two byte field that is available for future use
- NEGATIVE WORD COUNT: The negative (two's complement) of the total number of words remaining in the VIDS data block, including the block check word.
- BLOCK SEQUENCE NUMBER: A two byte incremental count identifying the VIDS data block.
- ACKNOWLEDGEMENT REQUEST BITS: A two byte field indicating, if the first byte is non-zero, that the error-free receipt of the VIDS data block must be acknowledged via the data link. The second byte is not used.
- ELAPSED TIME - MOST SIGNIFICANT BYTES: A two byte field holding the most significant bytes of a four byte word. The word contains the elapsed exercise time in milliseconds. This time should be incremental and representative of the data contained within the block.
- ELAPSED TIME - LEAST SIGNIFICANT BYTES: The least significant two bytes of the elapsed exercise time.

Figure 7-3 provides the type definition of the data structure implementing the VIDS data block header.

Following the header is the series of predefined VIDS messages. The types and order of the messages vary between VIDS data blocks; however, the

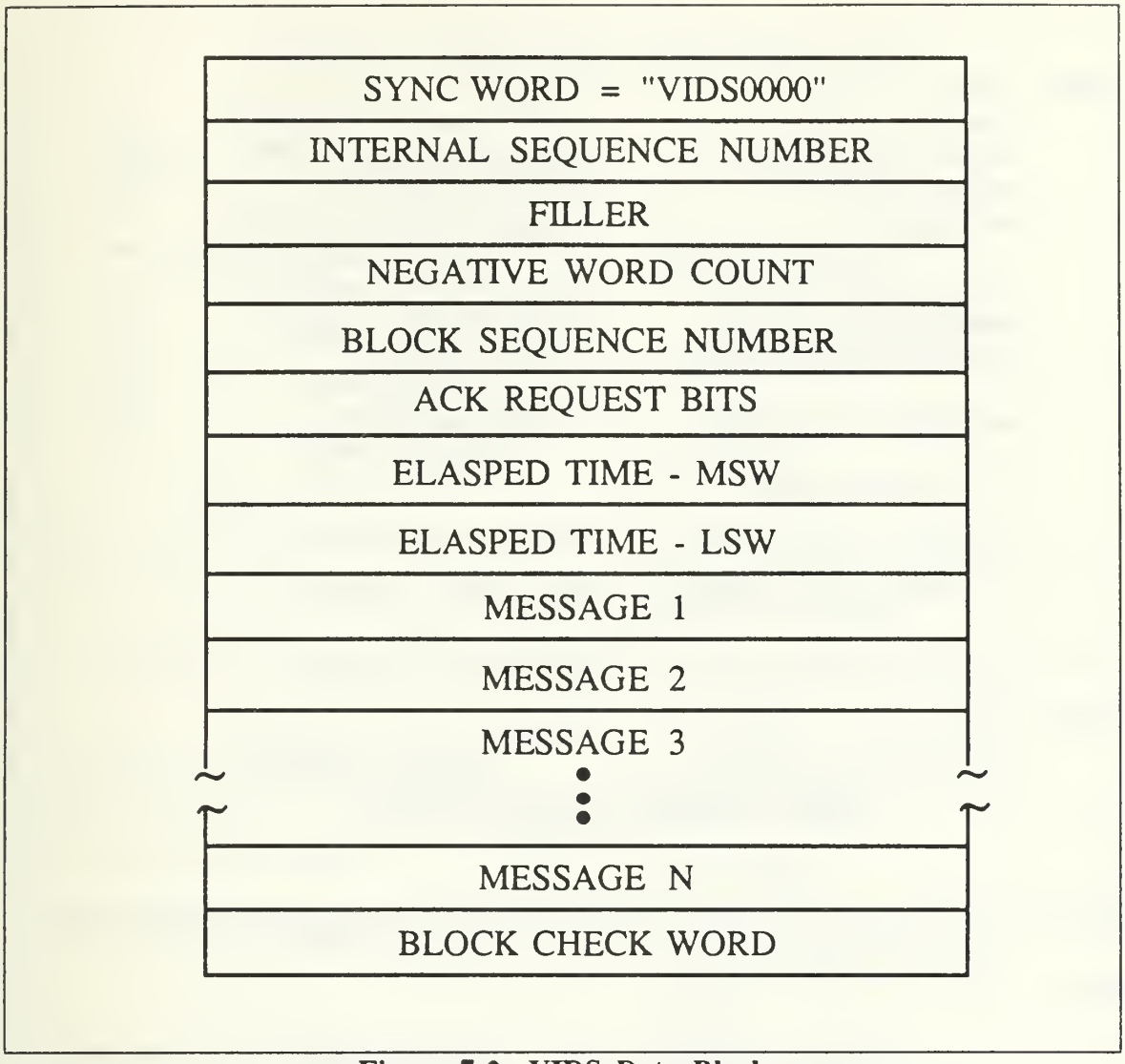


Figure 7-2 VIDS Data Block

messages must all come from the predefined set. The sum of all the messages cannot exceed 788 bytes, and a message may not be split between two VIDS data blocks. Figure 7-4 provides the type definition of the data structure implementing the VIDS messages, and a summary of the predefined messages is given in Table 7-2. For this application, the two most important messages are the type 1, *player position message*,

```
typedef  struct  {
         char sync_word[8];          /* ASCII header "VIDS0000".        */
         unsigned int inter_seq_no;  /* Internal sequence number.       */
         unsigned short filler;      /* Filler, (future use).           */
         short count;                /* Negative number of 2 byte       */
                                     /*       words remaining in block. */
         unsigned short block_seq;   /* Block sequence number.          */
         unsigned short ack_req;     /* Acknowledgement field.          */
         unsigned short msw_time;    /* Elapsed time most               */
                                     /*       significant word.         */
         unsigned short lsw_time;    /* Elapsed time least              */
                                     /*       significant word.         */
         }  vids_block_header;
```

Figure 7-3 VIDS Data Block Header Structure

```
typedef  struct  {
         unsigned short msg[394];    /* VIDS Messages Structure         */
```

Figure 7-4 VIDS Messages Structure

and the type 11, *start-time message*. They are described in greater detail later in this chapter.

The last component of the VIDS data block is the block check word. This two byte field is used to detect errors in transmission. The value of the field is the 16 bit frame check sequence generated by a cyclic redundancy check of the entire VIDS data block.

TABLE 7-2 PREDEFINED VIDS MESSAGES

TYPE	MESSAGE NAME	FUNCTION
1	Player Position Message	Specify player symbol and X, Y position
2	Player Annotate Message	Define annotation for specified player
3	Detail Data Update	Update player's detail data base
4	Graphic Element Message	Define graphic element for maintaining specified menu overlay list
5	Communication Message	Operator transmission of text data
6	Header Definition	Define contents of the display headers
7	None	Not defined
8	Symbol Definition	Initialize player symbols and other programmable fonts
9	Detail Template Definition	Define the formats of the detail data display
10	Menu Overlay Identification	Name and allocation storage for menu overlay lists
11	Start Time Message	Specify mission start time; signal to prepare for real-time operation
12	Color Index Message	Initialize the display's color tables

b. Between PROCESS_VDB and MPS II Modules

(1) Protocols. Like the interprocess communication link between the NETWORK_SIMULATOR and the PROCESS_VDB modules, the communication link between the PROCESS_VDB and MPS II modules uses the Internet protocols to establish the link. However, in this case the link uses the Internet *User Datagram Protocol/Internet Protocol* (UDP/IP) protocols. These are the same protocols as those used between networked MPS II modules. Since the PROCESS_VDB module functions as a source of platform data analogous to a networked MPS II module, it is essential that the protocols employed to communicate the data between the PROCESS_VDB and MPS II modules are the same.

(2) Data Structures. Like the interprocess communication link between networked MPS II modules, communication between the PROCESS_VDB and MPS II modules use a predefined set of string messages to communicate data between modules. An example of each type of message is given in Figure 7-5. For those messages that contain additional data, a description of the message body is provided in Table 7-3. The predefined messages the PROCESS_VDB module can send to the MPS II module are:

"** PROCESS VIDS DATA BLOCK MODULE initial msg ****"

PROCESS_VDB Initialization Message

"## PROCESS VIDS DATA BLOCK MODULE answer message #10000"

PROCESS_VDB Answer Message

"@@ PROCESS VIDS DATA BLOCK MODULE update message @10000.....
.....4.....48766.00000.....69577.00000.....14.705441
.....0.00000.....54.688786.....88000.00000.....23"

Update Message

"== PROCESS VIDS DATA BLOCK MODULE end message ====10000"

PROCESS_VDB End Message

"@@ MOVING PLATFORM SIMULATOR II initial message **"

MPS II Initialization Message

"## MOVING PLATFORM SIMULATOR II answer message ###10000"

MPS II Answer Message

"<< MOVING PLATFORM SIMULATOR II pvdb kill msg <<<<10000"

PROCESS_VDB Kill Message

Figure 7-5 Example Messages

TABLE 7-3 MESSAGE BODY DEFINITIONS

MESSAGE TYPE	FIELDS CONTENTS	DATA TYPE	FIELD WIDTH
Answer Message	Base Id Number	int	20
Update Message	Id Number	int	20
	Platform Type	int	20
	X Position	float	20
	Y Position	float	20
	Velocity	float	20
	Altitude	float	20
	Direction	float	20
	Time	float	20
	Message Count	long	20
PROCESS_VDB End Message	Base Id Number	int	20
PROCESS_VDB Kill Message	Base Id Number	int	20

- Initialization message.
- Answer message.
- Update message.
- PROCESS_VDB end message.

The predefined messages MPS II can send to the PROCESS_VDB module are:

- Initialization Message.
- Answer Message.
- PROCESS_VDB Kill Message.

For a discussion of the **initialization message, answer message** and **update message** see Chapter VI. The **PROCESS_VDB kill message** functions to instruct the

PROCESS_VDB module to terminate. After receiving a kill message and before terminating, the PROCESS_VDB module sends a **PROCESS_VDB end message**. The end message functions as a signal to all other running processes that the PROCESS_VDB process is terminating. The MPS II module uses this information to display the status of the PROCESS_VDB module.

B. THE PROCESS_VDB MODULE

Immediately upon start-up, the PROCESS_VDB module performs an initialization routine to establish, the network connections discussed earlier in this chapter. Once the connections are established, the module proceeds to retrieve the first VIDS data block. From this point on, the module continuously performs a sequence of tasks in a loop. This sequence is repeated for each VIDS data block the process receives until either the module is instructed to terminate operations, or when the source of the VIDS data blocks closes the connection that provides the VIDS data blocks. Two reasons the source can close the connection is either the end of the test exercise has occurred or the NETWORK_SIMULATOR has exhausted its prerecorded data. The ordered sequence of tasks the PROCESS_VDB module must accomplish for each VIDS data block is:

- Retrieve VIDS data block.
- Extract the time stamp from the header of the VIDS data block.
- Extract the length of the data block from the header.
- Search the VIDS message blocks looking for start-time messages and player position messages. When either is found, process the message data.
- Check for messages from MPS II.

1. Retrieving The VIDS Data Block

With the network connections already established, to read the VIDS data block through the network, a series of calls to the system function *read* is made. The read function is set to block when there is no data to read. Since between every VIDS data block there is at least two bytes of junk filler data, a search algorithm must first be used to read through the filler and locate the start of the VIDS data block. This algorithm accomplishes the task by reading two bytes at a time through the LAN until the characters "V" and "I" are read signaling the start of the VIDS data block (see Figure 7-6). After locating the start of the VIDS data block, 14 more bytes are read. This second read reads the VIDS data block up to the field containing the negative count of remaining bytes in the block. Finally, with this figure, the rest of the block can be read with one more read statement.

2. The VIDS Data Block Time Stamp

The header of each VIDS data block contains two fields that hold the elapsed time from the start of the exercise until the release of the VIDS data block. To determine the elapsed time, the two fields must be combined. This is a simple process that can be done many ways. The method chosen here is to shift the most significant two bytes to the left by 16, and then add the result to the least significant word. If the VIDS data block contains any player position messages, the elapsed time is used to calculate the platform's velocity.

```
gotheader = FALSE;
ateof = FALSE;

/* Loop until the end of the data or a VIDS data block is found. */
while (!gotheader)
{

   /* Read next two bytes, if there is only one its the end of the file. */
   if ((byte_cnt = read(network_socket, &vdb, 2)) < 2)
   {
      ateof = TRUE;
      printf("End of file.\n");
   }

   /* Test for start of a VIDS data block.  It starts with a "VI". */
   if ((vdb.sync_word[0] == 'V') && (vdb.sync_word[1] == 'I')
         && (!ateof))
      gotheader = TRUE;

}  /* End of while loop that looks for start of a VIDS data block. */
```

Figure 7-6 Algorithm to Locate Start of VIDS Data Block

3. The VIDS Data Block Length

The header also contains the length of the VIDS data block. Since the length of a VIDS data block varies with the number and type of VIDS messages the data block contains, the length is needed to determine where the VIDS data block ends. The length is stored as the negative count of two byte words remaining in the VIDS data block. The length data is used twice. First, when initially reading the data through the LAN and, second, when processing the VIDS messages. When reading the data through the network, the procedure is to first read the data up to the length field and then using the count of the remaining number of bytes, read the rest of the block. To get the total number of bytes that comprise all the VIDS messages, multiply the

count by minus two and subtract 14. The 14 accounts for the bytes in the acknowledgement and time fields which are between the count field and the start of the VIDS messages and the block check word, which is after the VIDS messages.

4. The VIDS Messages

The only VIDS messages we are concerned about are the start-time messages and player position messages. These are, as are the types of all VIDS messages, easy to discern. The first two fields in every VIDS message block have a fixed structure and provide the information needed to parse the VIDS data block, and separate the messages of interest from those that are not. The first is the length of that message, and the second field is the type of the message. A start time message is a type 11 message, and a player position message is a type 1 message. However, extracting the information in the correct form from the data structure is not simple. The data type for the count and the type field in their respective data structures is an unsigned char. But, in the VIDS messages they are stored together as an unsigned short. Remembering that an unsigned short is two bytes and an unsigned char is one byte, the count can be obtained by shifting the value in the short data structure to the right 8 bits. The message type can be obtained by masking out the upper byte.

a. The Start-Time Message

The start-time message specifies the time the exercise started and serves as a signal that real-time operations have started. The seven fields that constitute the message format are:

- LENGTH OF MESSAGE: The number of two byte words in the message. (All start time messages are 12 bytes long)

- TYPE OF MESSAGE: Field identifying the type of message. (A start time message is a type 11 message)
- DAY OF EXERCISE: Day of the year the exercise started expressed as a number between 0 and 366, inclusive.
- HOUR EXERCISE STARTED: Hour of the day the exercise started expressed using a 24 hour clock.
- MINUTE EXERCISE STARTED: Minute of the day the exercise started expressed as a number between 0 and 59, inclusive.
- SECOND EXERCISE STARTED: Second of the day the exercise started expressed as a number between 0 and 59, inclusive.
- MILLISECOND EXERCISE STARTED: Millisecond of the day the exercise started expressed as a number between 0 and 999, inclusive.

Figure 7-7 provides the type definition of the data structure implementing the start-time message.

```
typedef  struct  {
         unsigned char count;          /* Number of 16 bits in msg.        */
         unsigned char type;           /* The type field for this message. */
         unsigned char day;            /* Day of the year of exercise.     */
         unsigned short hour;          /* Hour the exercise started.       */
         unsigned short min;           /* Minute the exercise started.     */
         unsigned sec;                 /* Second the exercise started.     */
         unsigned short mil;           /* Millisecond exercise started.    */
         }  start_time_message;
```

Figure 7-7 Start Time Message Structure

b. The Player Position Message

The player position message contains eight fields and is ten bytes long. The fields are:

- LENGTH OF MESSAGE: The number of two byte words in the message. (All player position messages are 10 bytes long)
- TYPE OF MESSAGE: Field identifying the type of message. (A player position message is a type 1 message)
- PLATFORM IDENTIFICATION NUMBER: The identification number of the platform this message is providing information about.
- FILLER: Unused.
- PLATFORM SYMBOL: Used in FHL visual information display system to set the desired ASCII character to display the platform.
- PLATFORM COLOR AND FONT: Used in FHL visual information display system to set the color and font of the character used to display the platform.
- X POSITION: The X position of the platform in UTM grid coordinates with an offset.
- Y POSITION: The Y position of the platform in UTM grid coordinates with an offset.

Interest is in the third, seventh, and eighth fields in the player position message. The third field provides the platform's identification number. The number uniquely identifies the platform, and is assigned to the platform prior to the exercise. The seventh and eighth fields give the position of a platform at the time recorded in the header block. This position is in UTM grid coordinates with an offset of -40,000 in the X and Y directions. The offset is required because the fields in the player position message data structure used to record the X and Y position are, for historical reasons, unsigned short integers. The problem that necessitates the offset rises because most implementations of the C programming language provide only two bytes for short integers. This makes the maximum limit for either the X or Y position 65,536. Since the accuracy of the position data is to the nearest meter, and since it takes five digits to specify the x or y position to the nearest meter, any UTM grid coordinate greater than 65,536 could not be recorded. The coordinates of the majority of the FHL exercise area exceed this limit. Another limitation of the player position message structure is it does not provide a field for altitude. This is a serious limitation as

another goal of CDEC is to graphically display intervisibility data. If the player is an aircraft or other flying object, there is no way to determine whether or not it is visible because it is flying high enough to not be blocked from view by the terrain. Figure 7-8 provides the type definition of the data structure implementing the player position message.

```
typedef  struct  {
         unsigned char count;          /* Number of 16 bit words in msg.  */
         unsigned char type;           /* The type field for this message.  */
         unsigned char id;             /* The player's id number.           */
         unsigned char filler;         /* Unused.                           */
         unsigned char symbol;         /* Desired ASCII char for symbol.  */
         unsigned char color_font;     /* The upper nibble is the color     */
                                       /* index of the symbol, the last     */
                                       /* two bits are the font size.       */
         unsigned short x_posn;        /* The x position of the platform.   */
         unsigned short y_posn;        /* The y position of the platform.   */
         }  player_position_message;
```

Figure 7-8 Player Position Message Structure

Once the PROCESS_VDB module has identified the message as a player position message, it must determine if an update message needs to be sent to MPS II. The process of determining if an update message must be sent is complex. However, by identifying as early as possible if an update message should or should not be sent, unnecessary processing is avoided.

(1) Determination To Send Update Message. A set of rules is used to determine if an update message should be sent. First, there are two times when an update message should never be sent. They are:

- Don't send an update message if this is the first player position message received for this platform.
- Don't send an update message if the platform was not selected as a platform to display.

An update message should never be sent for the first message because the velocity of the platform, as reported in the message, must be calculated by dividing the movement of the platform by the elapsed time during which it moved. However, the movement is calculated by finding the difference between the platform's last reported position and its current position, and in the case of the first player position message, there is no recorded last position. Also, an update message should not be sent if the platform was not selected as a platform to display. The procedure to select the platforms to display is provided in the user interface chapter (see Chapter IX).

In those cases where the first set of rules does not prevent the sending of an update message, a second set of rules is applied. These rules are based upon a series of thresholds that are defined in the file "process_vdb.h." The use of the thresholds was designed to guarantee the periodic sending of update messages to insure the display of the platforms is kept accurate, yet prevent the sending of unnecessary messages that slow down MPS II's display rate. The second set of rules is:

- Send an update message if the change in velocity exceeds a defined velocity threshold.
- Send an update message if the elapsed time since the last update exceeds an elapsed time threshold.

- Send an update message if an update has not been sent for a defined number of player position messages for this vehicle.

By setting the velocity threshold to a number greater than zero, minor velocity changes caused by the imprecision in the position reporting instrumentation will be filtered. The position data are reported as accurate to the nearest meter. However, this means that data can be rounded off as much as half a meter. Assuming the platform's position is reported every second, which is a realistic figure, the imprecision will cause the platform's speed to vary by as much as 1.8 kilometers per hour. In the situation where the velocity actually did change, the thresholds that specify the maximum elapsed time between update messages, or the maximum of updates that will be prevented from being sent, will insure an update is eventually sent.

(2) Actions Taken When Update Must Be Sent. If the determination was made that an update message must be sent, a sequence of tasks must be followed. The in order sequence is:

- Calculate the vector velocity.
- Send the update message.
- Save the current platform X and Y position and the VIDS data block time.
- Reset the count of messages not sent to zero.

To minimize the calculations needed to compare the platform velocity against the velocity threshold, the X and Y components of the vector velocity are used. Only if the thresholds are exceeded, and an update message must be sent, is the vector velocity calculated. Calculating the vector velocity requires finding the magnitude and direction of the sum of the components. This is expensive in terms of CPU time.

With the communication links discussed earlier in this chapter established upon start up of the module, all that is needed to send a message is to compose the message and send it. To assemble the message, the system function *sprintf()* is used. The sprintf function creates a character string containing the update message data. This string is then concatenated to the end of a character string containing the message header to complete the full message. To send the message the system function *sendto* is used.

The next step is to save the platform's X and Y position and velocity and the VIDS block time. The next time a player position message is received for the platform, the X and Y position will be needed to determine the platform's movement during the elapsed time. The velocity will be used to determine if the velocity changes between receipt of the two player position messages. Finally, the VIDS data block time will be needed to calculate the elapsed time between the messages.

The last step taken after sending an update message is to reset the count of messages not sent to zero. As discussed earlier, this counter is used to insure messages are periodically sent to maintain the accuracy of the display. The count is incremented every time a message is not sent for a platform. Whenever this count exceeds a threshold, an update message will be sent regardless of the platform's movement and velocity changes.

(3) Actions Taken When Update Is Not Sent. A different sequence of tasks is followed for the case an update message is not sent. The shorter sequence is:

- Save the current platform X and Y position and VIDS data block time
- Increment the count of messages not sent

Even though the current position of the platform was not sent, its new position must be saved, along with the VIDS data block time. If the information is not saved and a change in the velocity occurred between this and the next player position message, the velocity change would be averaged over the elapsed time since the last update resulted in the incorrect reporting of the new velocity.

Finally, the count of update messages not sent must be incremented by one.

C. THE NETWORK_SIMULATOR MODULE

Like the PROCESS_VDB module, the NETWORK_SIMULATOR module performs an initialization routine to establish its network connections. However, in this case, there are two additional requirements of the initialization routine. The first requirement is to open the file containing the prerecorded exercise data that is used to load the department's LAN. Early in the development of the NETWORK_SIMULATOR module, a decision was made to use data read from a file rather than artificially generating the data on the spot. Although this approach limits the duration of the simulation to the amount of data available in the file and requires maintaining large files containing the exercise data, it allows the replaying of actual exercise data. During an average exercise, one VIDS data block is generated every second resulting in a data file of approximately one megabyte for a 50 minute

simulation. The second additional requirement is to record the start time of the simulator. The start time is used to time the release of the prerecorded exercise data into the LAN. After performing the initialization routine, the network simulator module performs the following sequence of tasks until instructed to terminate:

- Get VIDS data block from data file.
- Get time stamp of VIDS data block and wait until elapsed time passes.
- Release VIDS data block into LAN.

1. Reading the VIDS Data Block

The procedure the NETWORK_SIMULATOR uses to read the VIDS data blocks from the prerecorded data file is identical to the procedure the PROCESS_VDB module uses to read the VIDS data block through the network. The reason for this similarity is the *read* system function. The *read* function can be used to read from either a file or through a network socket. In the case of the network function, the read function can only be used when the socket is connected, which is the case for the connection between the NETWORK_SIMULATOR and the PROCESS_VDB module [Ref. 15].

2. Waiting for Elapsed Time

The procedure the NETWORK_SIMULATOR uses to extract the time stamp and determine the elapsed time from the start of the test exercise until the VIDS block is to be released is also identical to the procedure the PROCESS_VDB module uses. Once the VIDS data block elapsed time is determined, the NETWORK_SIMULATOR gets the actual time and subtracts it from the start time recorded during the initialization process to find the elapsed time since start of the simulation. If the

simulation elapsed time is greater than the elapsed time recorded in the VIDS data block, the release of the VIDS data block is late. In this case, the VIDS data block is immediately released, and an error message is printed. Otherwise, the process enters a delay loop until the elapsed time since the start of the simulation equals the elapsed time of the VIDS data block. At this point, the entire VIDS data block is released into the LAN.

3. Releasing the VIDS Data Block

To release the VIDS data block into the network, the system *write* function is used. Like the system *read* function, the *write* function can be used to write data to either a file or a network socket if the socket is connected. In addition to releasing the VIDS data block into the network, the junk filler bytes between the VIDS data blocks must be released into the network to simulate this aspect of the communication link. The junk filler bytes are released into the network as soon as they are identified as filler bytes and not at the start of the next VIDS data block.

D. The Moving Platform Simulator II Interface

The MPS II treats the platform update messages it receives from the PROCESS_VDB module identically to update messages it receives from other running MPS II processes. A discussion of MPS II's handling of update messages is provided in Chapter V.

VIII. INTERVISIBILITY CALCULATIONS AND DISPLAY

A. SYSTEM ARCHITECTURE

1. Overview

Another objective of CDEC is to be able to graphically display intervisibility data. Given the location of an observer platform, CDEC wanted a display to show when the observer can visually see a specified target platform. Both the observer and the target can be moving. This intervisibility display capability is needed to analyze the performance and effectiveness of the FOGM in comparison with the Tube-launched, Optically-tracked, Wire-guided (TOW) missile. The FOGM operator does not have to visually see a target to engage it. However, with the TOW missile, the operator must have an unobstructed line-of-sight to engage the target.

To provide this capability, another new module was developed to work with MPS II. Like the modules that support MPS II's ability to graphically display actual test exercise platforms (see Chapter VII), this module is designed as a separate module to distribute the processing. Additionally, as a separate module, it can independently serve more than one MPS II and any platforms generated by the PROCESS_VDB module. This new module is called the PROCESS_LOS module because the process checks the Line-Of-Sight (LOS) between the observer and the target platforms. This is done by comparing the elevation of the ground against the elevation of an imaginary line from the observer to the target. If, at a specified point, the elevation of the

ground is greater than the elevation of the imaginary line at that point, the observer's view of the target is blocked.

The PROCESS_LOS module receives the platform data from both the PROCESS_VDB and MPS II modules. Any one of the platforms can be selected as the observer, and any combination of the remaining can be selected as targets. Permitting the selection of multiple targets provides the capability to simultaneously determine the intervisibility between an observer and many target platforms. Additionally, the observer can be placed at a fixed location.

The PROCESS_LOS module runs in one of two modes. In the first mode, the process can graphically display the intervisibility determinations itself, and run independently of MPS II by receiving all required platform data from the PROCESS_VDB module. In the second mode, the process runs without a graphic display and may be run as a background process detached from the terminal. In both modes the process sends MPS II formatted messages containing concise drawing instructions, which allows MPS II to also display the intervisibility data.

2. Interprocess Communication

Like the modules needed to display actual exercise platforms in real time, the display of platform intervisibility depends on interprocess communication links. In this case the links are to the PROCESS_VDB and MPS II modules. Figure 8-1 provides an overview of the connections to the LAN and the interprocess communications links between the modules.

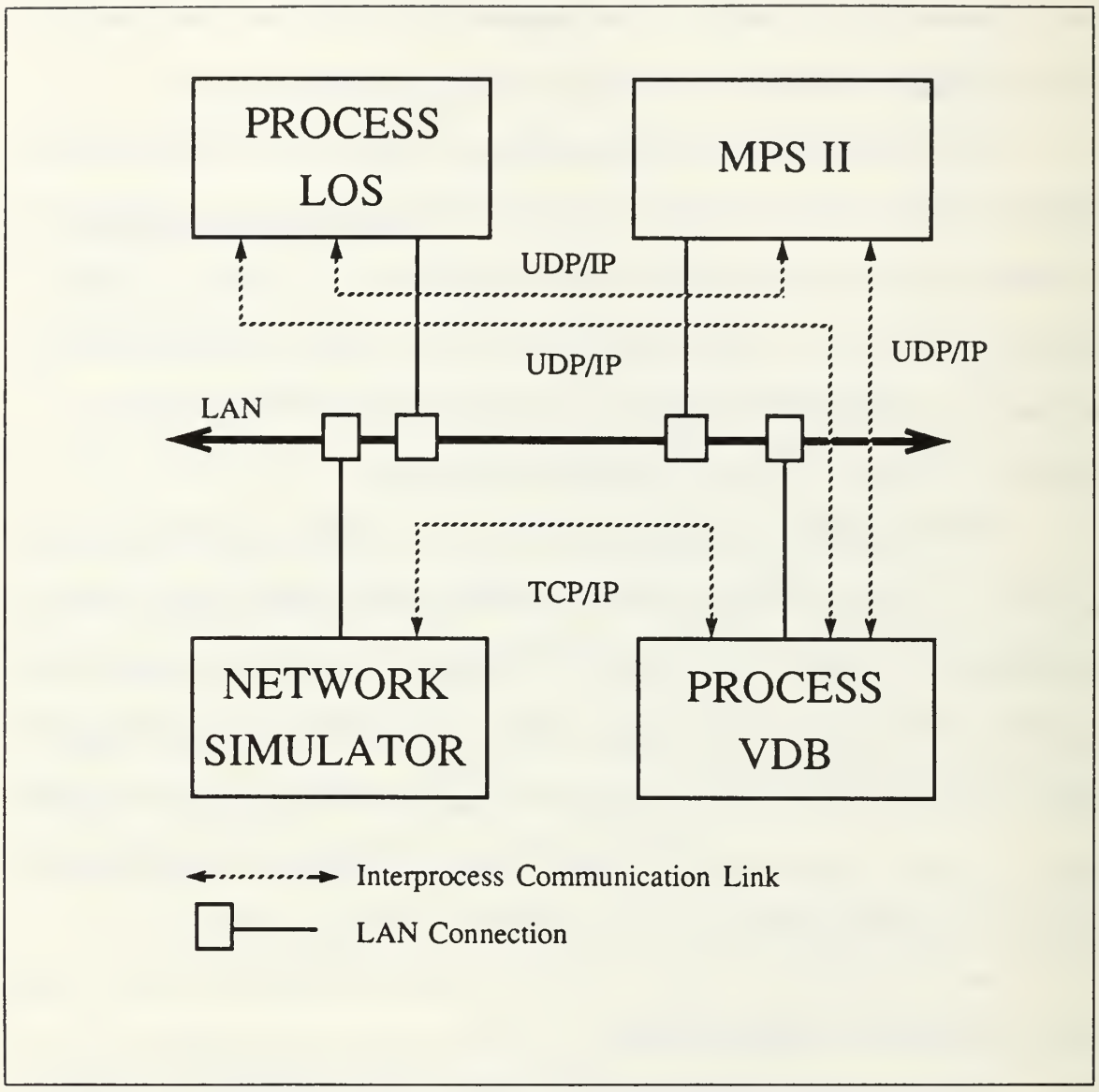


Figure 8-1 Interprocess Communications Links

a. Protocols

Since the PROCESS_LOS module gets the platform data it uses to determine the locations of the observer and target platforms from running PROCESS_VDB and MPS II modules, the protocols it uses are identical to those used between the PROCESS_VDB and MPS II modules (see Chapters VI and VII). The connectionless broadcast nature of the link also allows the PROCESS_LOS module to

receive the messages sent between PROCESS_VDB and MPS II modules and between networked MPS II modules. This receipt of the messages by the PROCESS_LOS module is entirely transparent to the other modules.

b. Data Structures

In addition to using the same protocols, the PROCESS_LOS module also uses some of the same messages and formats used to communicate between the PROCESS_VDB and MPS II modules and between networked MPS II modules (see Chapters VI and VII). These messages are augmented by an additional set of messages.

There are two messages the PROCESS_LOS shares with the PROCESS_VDB and MPS II modules. The two messages are:

- Update Message
- PROCESS_VDB Kill Message

The PROCESS_LOS module never sends an **update message.** However, it listens for and receives all the update messages sent by the PROCESS_VDB and MPS II modules. They are the module's sole source of platform data. The **PROCESS_VDB kill message** is also used by the PROCESS_LOS module. Like the MPS II module, the PROCESS_LOS module is able to start and terminate the PROCESS_VDB module. A detailed discussion of the structure of and use of both messages is given in Chapter VII.

The additional messages are used to communicate intervisibility data to MPS II. Consistent with the format of all the other interprocess messages, the additional messages are also comprised of character strings and contain a 50 character

header followed by data fields. An example of each message is given in Figure 8-2. A description of each message body is provided in Table 8-1. The additional messages are:

- Observer LOS message
- Target platform intervisibility message
- Observer position message
- PROCESS_LOS kill message
- PROCESS_LOS end message

An **observer LOS message** is used to pass platform intervisibility data to MPS II. A message is sent every time the intervisibility between the observer and a target is determined. The message provides the observer and target platform's position, if the observer can see the target, and if the observer's view is blocked, the point of the obstruction. The message consists of the following eight fields:

- Target platform's unique identification number.
- X position of the observer platform.
- Y position of the observer platform.
- X position of the target platform.
- Y position of the target platform.
- X position of the obstruction if any, otherwise zero.
- Y position of the obstruction if any, otherwise zero.
- Number two if the observer can see the target, otherwise one.

If the observer's view is obstructed, the first point of obstruction along the observer's line of sight to the target is reported. The numbers used to indicate the platform intervisibility are derived from the system defined manifest constants for GREEN and RED. In the system include file "gl.h", the manifest constant GREEN is defined to be two, and RED is defined to be one. If the observer can see the target, the platform intervisibility is reported as GREEN. If the observer's view, is blocked the intervisibility is reported as RED.

"@@ PROCESS VIDS DATA BLOCK MODULE update message @10000.....
.....4.....48766.00000.....69577.00000.....14.705441
.....0.00000.....54.688786.....88000.00000.....23"

Update Message

"<< PROCESS_LOS process_vdb kill message <<<<<<<<<<10000"

PROCESS_VDB Kill Message

"++ PROCESS_LOS observer line of sight message ++++"
10022.....67858.5675.....45555.3954.....78654
....88957.....69594.....98595.....1"

Observer LOS Message

"|| PROCESS_LOS tgt platform intervisibility msg ||"
10014.....68955....74899.....66564.....78895.....1"

Target Platform Intervisibility Message

">> PROCESS_LOS observer position message >>>>>>>>>"
1.....67858.5675.....45655.3954.....
35.43.....54.3543.....274.1253"

Observer Position Message

"// MOVING PLATFORM SIMULATOR II plos kill message/10000"

PROCESS_LOS Kill Message

"]] PROCESS_LOS process end message]]]]]]]]]]]]]]"

PROCESS_LOS End Message

Figure 8-2 Example Messages

TABLE 8-1 MESSAGE BODY DEFINITIONS

MESSAGE TYPE	FIELDS CONTENTS	DATA TYPE	FIELD WIDTH
Observer LOS Message	Platform ID Num	int	10
	Observer X position	float	20
	Observer Y position	float	20
	Target X position	long	10
	Target Y position	long	10
	Obstruction X position	long	10
	Obstruction Y position	long	10
	Intervisibility Determination	int	10
Target Platform Intervisibility Message	Platform ID Num	int	10
	Previous Target X position	long	10
	Previous Target Y position	long	10
	Current Target X position	long	10
	Current Target Y position	long	10
	Intervisibility Determination	int	10
Observer Position Message	Platform Type	int	10
	Observer X position	float	20
	Observer Y position	float	20
	Observer Velocity	float	20
	Observer Altitude	float	20
	Observer Course	float	20
PROCESS_LOS Kill Message	Base Identification Number	int	20

A **target platform intervisibility message** is also used to pass platform intervisibility data to MPS II. In this case, the data is specific to the target platform.

Although the data is a duplication of data contained in the observer LOS messages and can be acquired from data contained in two consecutive messages, it is replicated to consolidate the data in one message to reduce the amount of processing and data storage needed to initially display or, when chosen, redisplay the data. The message contains the following six fields:

- Target platform's unique identification number.
- X position of the previous reported location of the target platform.
- Y position of the previous reported location of the target platform.
- X position of the target platform.
- Y position of the target platform.
- Number two if the platform is visible, otherwise one.

Again, the system defined constants for RED and GREEN are used to specify the intervisibility in the last field of the message.

The **observer position message** is used to communicate the observer's position between the PROCESS_VDB and MPS II modules. Since both modules can specify and display the position of the observer, the message is used by both modules. The seven fields in the message are:

- Type of observer platform (jeep, tank, stationary position, etc.).
- X position of the observer platform.
- Y position of the observer platform.
- Velocity of the observer platform.
- Altitude of the observer platform.
- Course in compass degrees of the observer platform.

The velocity, altitude, and course of the observer platform are included on the message to allow the dead reckoning scheme, employed by MPS II to move platforms between update messages, move the observer.

The **PROCESS_LOS kill message** serves to instruct the PROCESS_LOS module to terminate. After receiving a kill message and before terminating, the module sends a **PROCESS_LOS end message**. The end message serves as a signal the module is about to terminate. This information is used by MPS II to display the status of the PROCESS_LOS module.

B. DETERMINATION OF INTERVISIBILITY BETWEEN PLATFORMS

A sequence of tasks, referred to here as the intervisibility algorithm, is executed to determine the intervisibility between the observer and the target platforms. The sequence is repeated for each platform update message received through the LAN. To optimize the algorithm, a set of rules is used to determine if the entire sequence must be followed, or at what point in the algorithm are all the required steps for a given update message completed and the process can proceed onto the next target platform update message. Figure 8-3 details the sequence of tasks in the intervisibility algorithm.

1. Optimization Rules

To maximize the number of platform update messages and intervisibility determinations the PROCESS_LOS module can handle in time, a set of rules is employed to optimize the intervisibility algorithm. These rules are:

- Don't determine the intervisibility for the first platform update message received for a platform.
- If the observer is in a platform, don't determine the intervisibility for that platform.
- In addition to the points reported in the platform update messages, only determine the intervisibility at points a specified distance apart along the target platform's path.

Task Number	Description of Task
1	Get a target platform update message.
2	Determine the number of points along the target platform's path that will be checked. Determine the X and Y distance between these points.
3	For each point on the target's path, referred to here as the target point, do the following: (1) Determine the number of points, along the line-of-sight from the observer to the target point, that will be checked for an obstruction. These points will be referred to as the LOS points. (2) Determine difference in elevation between the observer's location and the target point. Calculate the change in elevation between each LOS point. (3) Step through each LOS point, one at a time, checking to see if it blocks the observer's view. If an obstruction is found, immediately stop and go to the next step. Otherwise, check the next LOS point. If all the LOS points have been checked, go to the next step. (4) If the PROCESS_LOS module is running in the display mode, display the results of this intervisibility check. (5) Save the results of this intervisibility check to a file for display at a later time. (6) Notify MPS II of the results of this intervisibility check by sending an observer LOS message and a target platform intervisibility message.

Figure 8-3 Intervisibility Algorithm

- Similarly, for the line-of-sight path from the observer to the target platform, only check for an obstruction at points a specified distance apart along this imaginary line.
- As soon as an obstruction is found in the observer's line-of-sight, move onto the next platform update message.

Because the intervisibility algorithm requires two positions for a platform, the intervisibility is never determined for the first platform update message for a target platform. Waiting for the receipt of two update messages for a target platform allows the algorithm to determine the intervisibility for points between the reported points in the update messages but still along the platform's path. This algorithm is described in greater detail later in this chapter.

The intervisibility also does not have to be determined for the update messages on a platform if the observer is in the platform. Obviously, if the observer is in the platform, he can see the platform.

To limit the number of times the intervisibility algorithm is applied to a target platforms's path, a target path segment length is specified. The intervisibility algorithm is always applied to the locations reported in the update messages. However, only if the distance between the reported points is greater than this length, are intermediate points along the path tested. A detailed discussion of the scheme used to determine the actual points to which the algorithm is applied is provided later in this chapter.

Likewise, the line-of-sight path between the observer and the target platform can be divided into segments. However, in this case the specified LOS segment length is employed and the points are checked for an obstruction in the observer's view.

Again, a detailed discussion of the scheme used to determine the actual points used to check for an obstruction is provided later in this chapter.

The last rule is to move onto the next update message as soon as an obstruction is found. CDEC's requirement is only to know if the observer can or cannot see the target. CDEC is not concerned with the actual location of the point of obstruction.

2. Determining the Elevation of a Point

a. Elevation Database

To ascertain the observer, target platform, and LOS point elevations needed to determine the intervisibility, the PROCESS_LOS module uses the same terrain database file as MPS II. A description of this file is provided in Chapter III. Like MPS II, only the elevation data is used, and the vegetation data is ignored since most of the codes indicate that the information is unknown [Ref. 1]. However, unlike MPS II, the PROCESS_LOS module reads and stores the entire database file. The database is stored in the two dimension array *elevation_data[X][Y]*. The first dimension corresponds to a normalized value of the X component of the UTM grid coordinate. The second dimension corresponds to a normalized value of the Y component of the UTM grid coordinate. The components are normalized to account for the resolution of the database and the offset from the origin of the FHL map. The lower left UTM grid coordinate of the terrain database has an X component of 41000 and a Y component of 60000. Figure 8-4 provides the algorithm used to normalize the values.

```
/* Algorithm to normalize the X and Y components of an UTM coordinate. */
x_normalized = (long)((x_position - 41000) / DATA_RESOLUTION);
y_normalized = (long)((y_position - 60000) / DATA_RESOLUTION);
```

Figure 8-4 Normalization Algorithm

b. Elevation Calculation

If the location of every observer, target platform, and LOS point corresponded exactly to a point in the database, the determination of their elevations would be easy. It would simply equal the value of the elevation_data array evaluated for the point. However, this is not the case. As a result, an algorithm must be used to calculate the elevation of positions located between the points in the database. The algorithm chosen here is to define a plane that contains the point, find the equation of the plane, and then solve for the elevation of the point.

(1) Defining The Plane. By assuming the point of unknown elevation lies in the plane defined by three adjacent points that are represented in the database and form a triangle around the point, its elevation can be calculated. This assumption permits a "best effort" approximation of the unknown elevation by interpolating the elevation from surrounding known points. The first step is to select three adjacent points. Figure 8-5 illustrates this procedure. Depending upon the points selected, it is possible to form two triangles around the point of unknown elevation. As long as the equation and points corresponding to the chosen triangle are used, it does not

matter which triangle is used. Figure 8-5 also shows both triangles for an example point. The triangle selected, for this application, is marked.

(2) Finding the Equation of the Plane. Next, using the elevation values of these three adjacent points, along with the known distance between the points, the equation of the plane that contains the points is derived. To derive the equation of the plane, first the normal to the plane is calculated. Then, using one of the known points in the plane, the actual equation of the plane is derived. An advantage of this approach is that it can be highly optimized for this situation. Since the X and Y distance between the points is equal for all points in the database, many of the intermediate values needed to calculate the equation of the plane and the elevation of the desired point in the plane reduce to one or zero. Because of the simplicity of the calculations in the algorithm, a decision was made to compute the values of the equation dynamically when needed rather during an initialization routine at start-up. This approach save considerable memory space.

(3) Solving the Equation for the Unknown Elevation. After deriving the equation of the plane, all that is left to complete the algorithm is to evaluate the equation by substituting in the known position of the point and solving for its unknown elevation. Figure 8-6 shows the algorithm, after the equations have been reduced and consolidated, used to calculate the elevation of an unknown point in an upper left triangle. Figure 8-7 shows the algorithm for the lower right triangle.

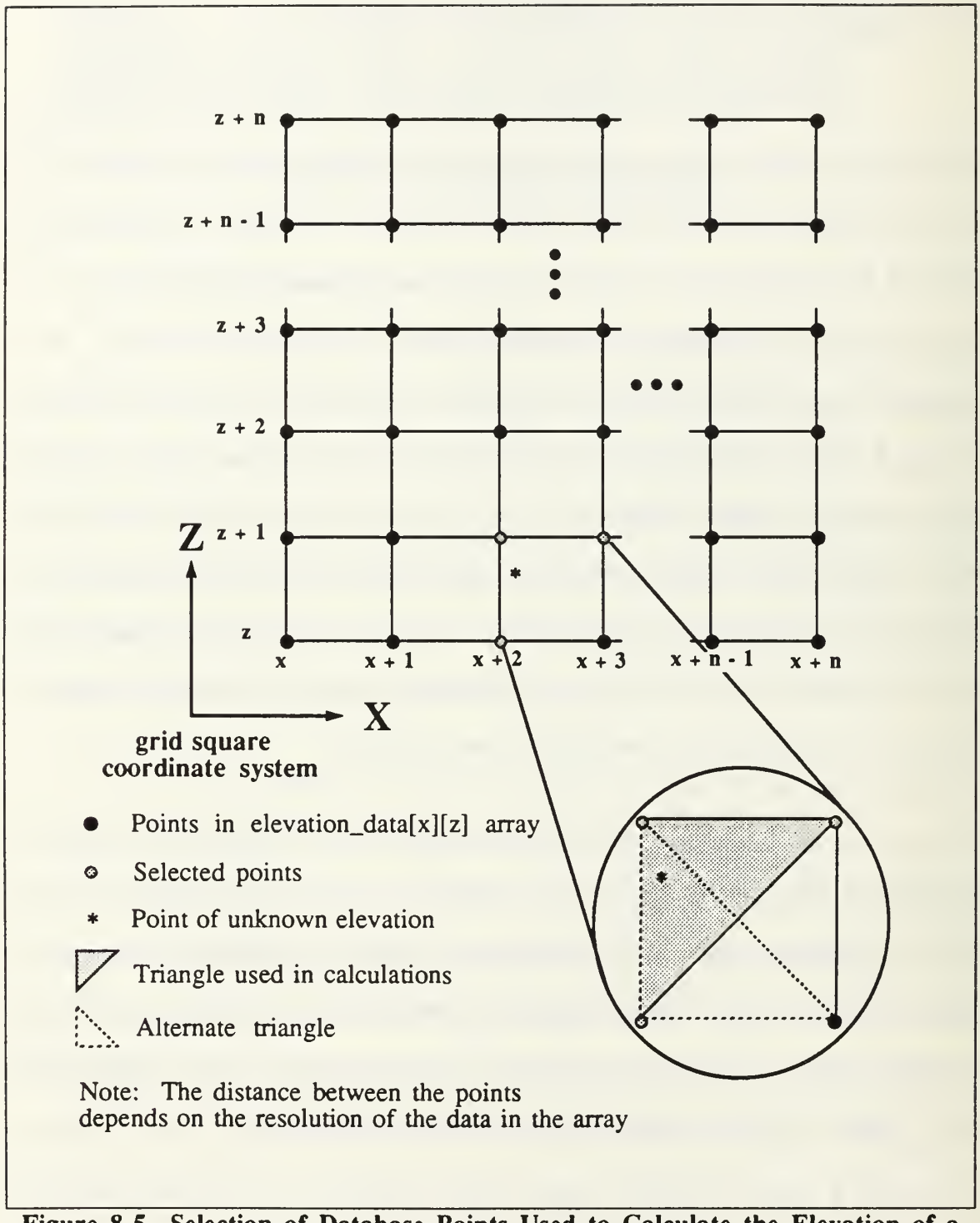


Figure 8-5 Selection of Database Points Used to Calculate the Elevation of a Position Not Represented in the Database

```
/* Normalize the coordinate of the point for which we need to find      */
/* the elevation.                                                       */
temp_x = (float)((x_position - 41000) / DATA_RESOLUTION);
temp_y = (float)((y_position - 60000) / DATA_RESOLUTION);
x_normalized = (long)temp_x;
y_normalized = (long)temp_y;


/* Find the offset of the point.  The offset is the percentage of       */
/* the distance that the point, for which we need to find the           */
/* elevation, is between the points in the database.  The lower         */
/* left corner of the triangle is the base point of the offset.         */
x_offset = (float)(temp_x - x_normalized);
y_offset = (float)(temp_y - y_normalized);


/* Find the elevation of the three adjacent points that form the        */
/* triangle around the unknown point.                                   */
elev_pt1 = elevation_data[ x_normalized ][ y_normalized ];
elev_pt2 = elevation_data[ x_normalized + 1 ][ y_normalized + 1 ];
elev_pt3 = elevation_data[ x_normalized ][ y_normalized + 1 ];


/* Finally, calculate the unknown elevation of the point.               */
unknown_elevation = ((elev_pt2 - elev_pt3) * x_offset)
                  + ((elev_pt3 - e_pt1) * y_offset) + elev_pt1;
```

Figure 8-6 Algorithm to Calculate Elevation of Point in Upper Left Triangle

3. Target and LOS Points

If the path distance a target platform travels between update messages exceeds a specified distance (the *veh_seg_length*), the path is divided into segments and the intervisibility between the observer and the target platform is determined at the end

```
/* Normalize the coordinate of the point.                          */
temp_x = (float)((x_position - 41000) / DATA_RESOLUTION);
temp_y = (float)((y_position - 60000) / DATA_RESOLUTION);
x_normalized = (long)temp_x;
y_normalized = (long)temp_y;


/* Find the offset of the point. The offset is the percentage of   */
/* the distance that the point, for which we need to find the      */
/* elevation, is between the points in the database. The lower     */
/* left corner of the triangle is the base point of the offset.    */
x_offset = (float)(temp_x - x_normalized);
y_offset = (float)(temp_y - y_normalized);


/* Find the elevation of the three adjacent points that form the   */
/* triangle around the unknown point.                              */
elev_pt1 = elevation_data[ x_normalized ][ y_normalized ];
elev_pt2 = elevation_data[ x_normalized + 1 ][ y_normalized ];
elev_pt3 = elevation_data[ x_normalized + 1 ][ y_normalized + 1 ];


/* Finally, calculate the unknown elevation of the point.          */
unknown_elevation = ((elev_pt2 - elev_pt1) * x_offset)
                  + ((elev_pt3 - e_pt2) * y_offset) + elev_pt1;
```

Figure 8-7 Algorithm to Calculate Elevation of Point in Lower Right Triangle

points of each segment. The end points are referred to as the *target points*. The number of target points, for a particular path, equals the path length divided by the value of veh_seg_length. The distance between the target points equals the distance the target platform traveled between the update messages divided by the number of target points. Theoretically, by reducing this number to an infinitesimally small number, the intervisibility between the observer and the target platform, as they move, can be continuously determined. However, technically this is not possible. A

reasonable figure would be to use the distance between the points in the database. This is the default value used by the PROCESS_LOS module. In calculating the distance between the target points, the x and y distances are used. As a result, the distance between target points is expressed in terms of an x and a y distance. This provides for the quick calculation of the location of each successive target point.

Similarly, the line-of-sight from the observer to the target platform is divided into segments. However, in this case, the end points of the segments, referred to as the *LOS_points*, are the points checked for obstructions in the observer's view. The distance used to calculate the number of LOS_points is the *los_seg_length.*

Both the veh_seg_length and the los_seg_length can be changed while the PROCESS_LOS module is running (see Chapter X). Figures 8-8 and 8-9 illustrate the segments, target points, and LOS points.

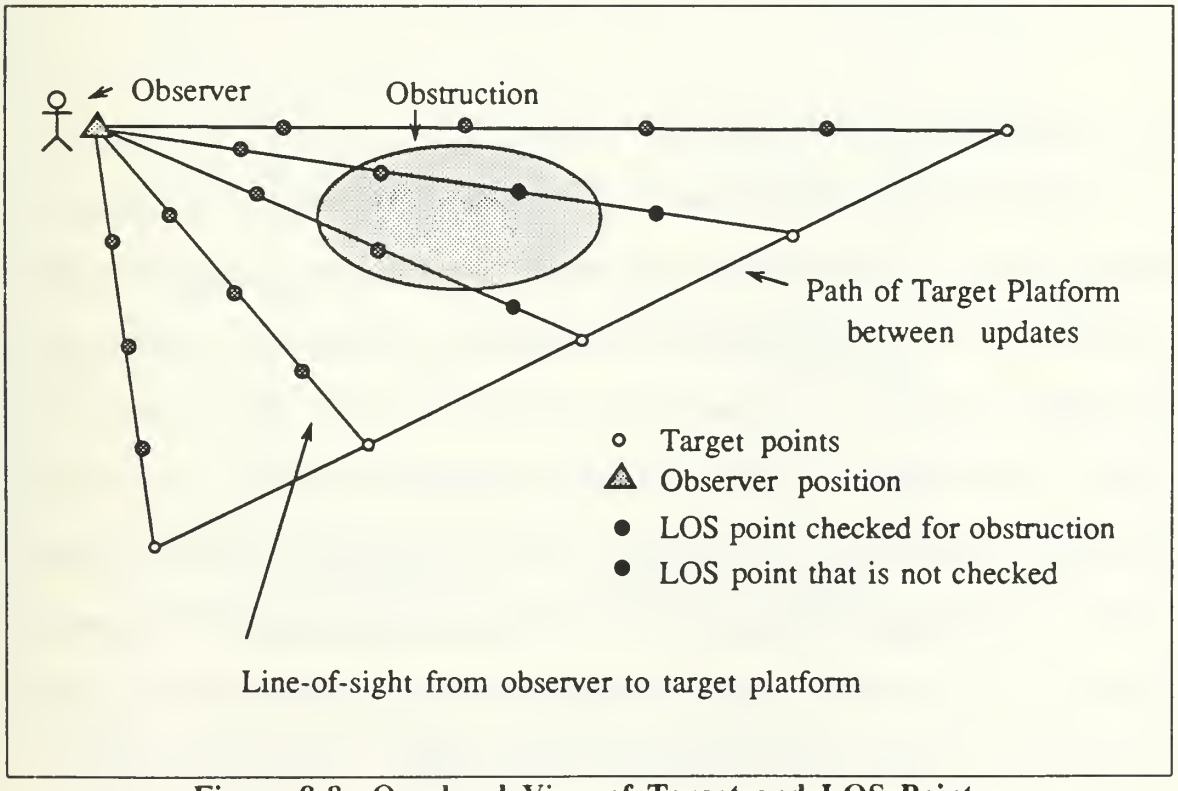


Figure 8-8 Overhead View of Target and LOS Points

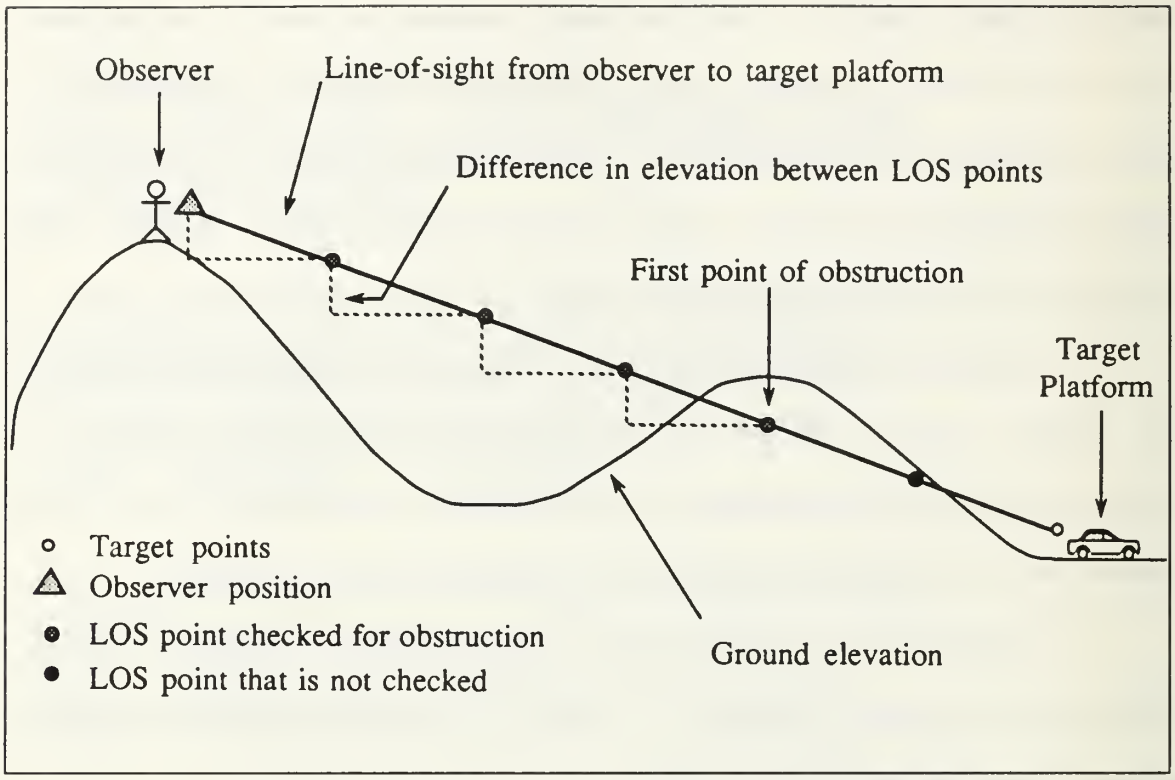


Figure 8-9 Side View of Target and LOS Points

C. PLATFORM INTERVISIBILITY DISPLAY

Although both the MPS II and the PROCESS_LOS modules are capable of displaying the results of the platform intervisibility determinations, the MPS II module has only a subset of options available on the PROCESS_LOS module. However, with both modules, the results are drawn on a two-dimensional map over the terrain the platforms are traveling. Additionally, both modules store the results of the intervisibility determinations for redisplay. The path the platforms traveled is drawn in green when the target platform is visible by the observer; otherwise the path is drawn in red. The observer's position is displayed using the standard military symbol for an observer: a triangle with a small dot in the center. To allow the observer to

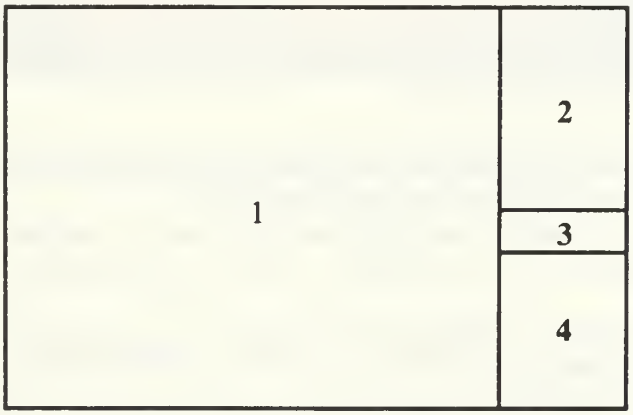
be repositioned without redrawing the entire map, the observer is drawn in the overlay mode.

1. PROCESS_LOS

a. Display Layout

The PROCESS_LOS module uses four graphics windows to display information. Figure 8-10 shows the relative positions of the windows. These windows are:

- Map window
- Statistics window
- Map data window
- Elevation data window



1. Map Window
2. Statistics Window
3. Map Data Window
4. Elevation Data Window

Figure 8-10 Window Layout

The **map window** is used to display the selected map and platform paths. To aid the actual display of the platform data, the *ortho2()* function is used to set the world coordinate system of the map display window equal to the UTM grid coordinate system of the map whenever platform data is drawn. As a result, the need to convert the UTM coordinates of the platform to some different display coordinate system is avoided.

The three remaining display windows are used to display supporting information such as the UTM grid coordinates of the lower left corner of the map, the elevation contour color scheme, the available and selected platforms, the position and elevation of the observer, the los_seg_length and veh_seg_length, and the platform number of the platform currently being processed.

b. Display Options

The PROCESS_LOS module has a comprehensive set of options to tailor the graphical display of intervisibility determinations. The display capabilities of the PROCESS_LOS module include:

- Ability to select any combination of available target platforms for display and to display their intervisibility determinations.
- Ability to clear the map of all platform data and, if desired, to select the same or a different set of platforms and to display their intervisibility determinations.
- Ability to position and to display the observer at any location on the map or in any available platform.
- If selected, display the line-of-sight from the observer and the location of the first obstruction if one exists for the selected platforms.
- Ability to adjust the length of the los_seg_length and the veh_seg_length.
- Ability to select the 35 x 35 kilometer map or any 10 x 10 or 1 x 1 kilometer map area from the 35 x 35 kilometer map area.
- Ability to adjust the terrain elevation color scheme.
- If selected, display the unique platform identification number of the target platform adjacent to its path.

- If selected, in addition to displaying the target points (the actual point where the intervisibility determination was made), display the target platform's path between the target points.
- Ability to display the current status of the process.

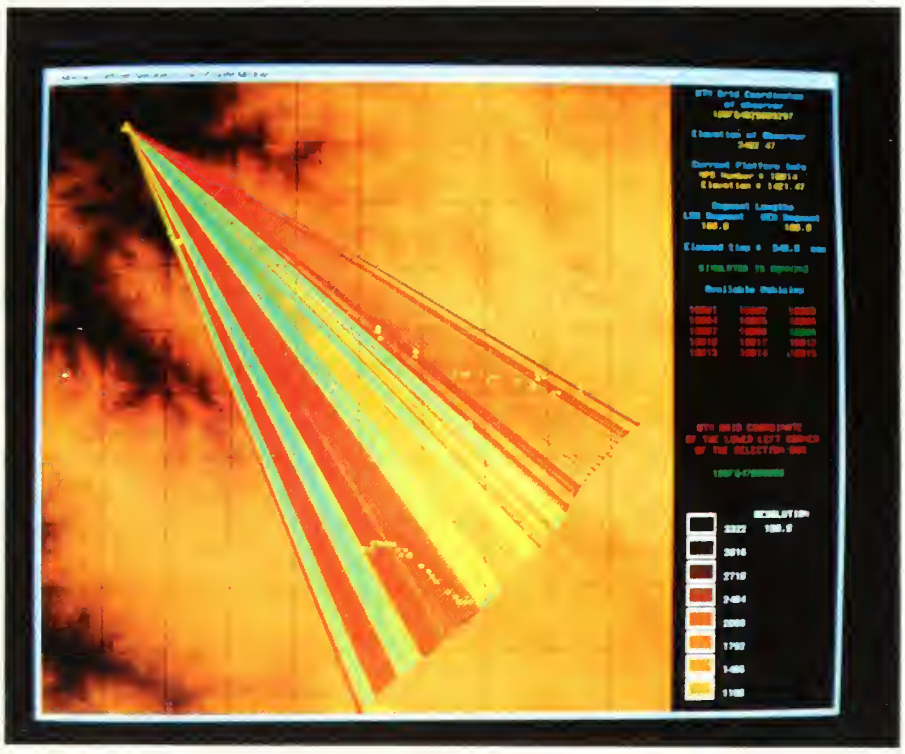


Figure 8-11 PROCESS_LOS Display of 10 x 10 Km Map With Line-Of-Sight Trace

Figure 8-11 shows the PROCESS_LOS display with the selection of the 10 x 10 kilometer map option and the drawing of the line-of-sight option turned on. Figure 8-12 also shows a 10 x 10 kilometer area, however, the drawing of the line-of-sight trace is turned off.

2. MPS II

MPS II uses a similar window layout, however only the map window is used. The other windows are used by other features of MPS II (see Chapter IV). Although not as comprehensive as the PROCESS_LOS module, MPS II has an ample set of available options to display the intervisibility determinations.

- Ability to select any combination of available target platforms for display and to display their intervisibility determinations.
- Ability to clear the map of all platform data and if desired, to select the same or a different set of platforms and to display their intervisibility determinations.
- Ability to position and to display the observer at any location on the map or in any available platform.
- If selected, display the line-of-sight from the observer and the location of the first obstruction if one exists for the selected platforms.
- Ability to adjust the terrain elevation color scheme.

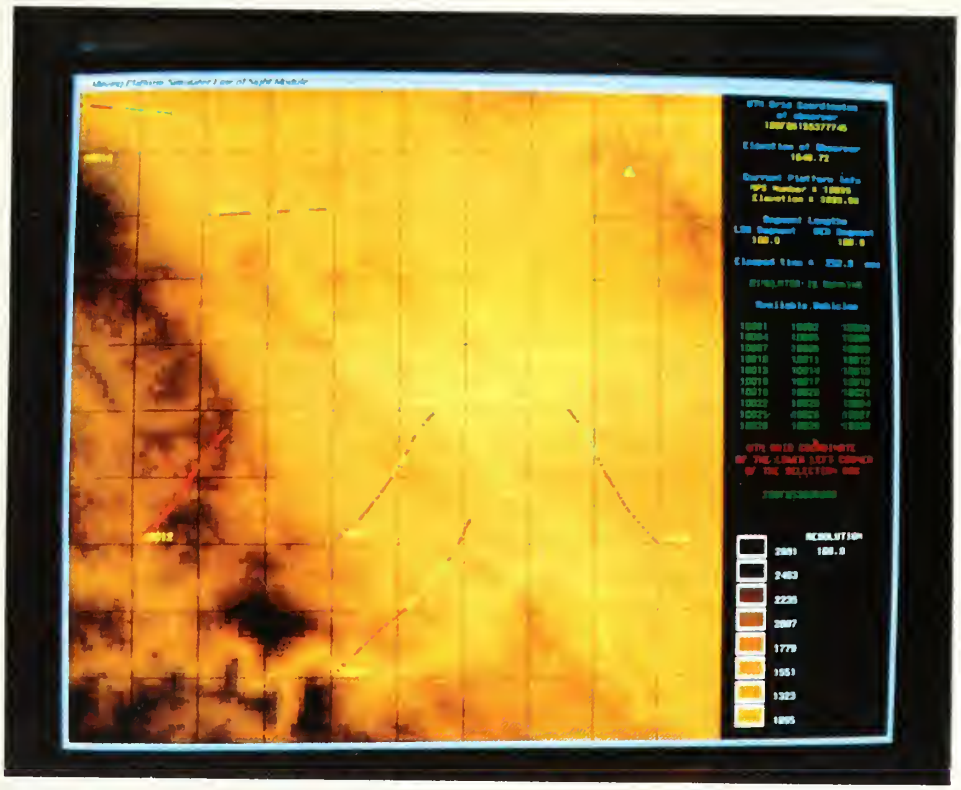


Figure 8-12 PROCESS_LOS Display of 10 x 10 Km Map Without Line-Of-Sight Trace

IX. SYSTEM EVALUATION OF MPS II

A. MOVING PLATFORM SIMULATION PERFORMANCE

There is ongoing research at the Naval Postgraduate School into the evaluation of high-performance graphics workstations. One of the objectives of this research was to continue that work. Silicon Graphics Inc. manufacturer's specifications state that the IRIS 4D/70GT is capable of producing 40,000 10x10 pixel, Z-buffered, lighted, Gouraud-shaded quadrilaterals per second. This number is established using optimized drawing code and does not reflect the performance that can be expected from a real-world application program. However, MPS, the predecessor of MPS II, does provide a more realistic reference to measure the performance of a graphics workstation. MPS is an application program of suitable complexity to provide a comparison between the manufacturer's claim of performance specifications and a real-world applications performance.

Using MPS's performance as a yardstick, MPS II was similarly evaluated. Two measurements are used to judge the performance of MPS II: **frames per second** and **polygons per second.** Frames per second is more important in the assessment of graphic simulators because it determines the quality of the display. Without a suitable frame rate, the user will not have the sense of smooth motion. However, the polygons per second measurement enables comparison to the manufacturer's specifications. A third measurement, **polygons per frame**, is used to judge the complexity of the display.

By improving the performance of MPS while increasing the complexity of the software, MPS II pushes the hardware to a greater extent than MPS to better evaluate what level of application program is suited for present day, off-the-shelf technology. It also gives a clue as to what type of applications will be possible with faster, more powerful hardware.

Both MPS and MPS II have a variety of complex routines which test the graphics workstations [Ref. 1:p. 61]. Computations completed during each iteration of the display loop include:

- Updating each platform's position and elevation
- Performing collision detection between platforms
- Performing platform coordinate transformation
- Calculating what terrain to draw
- Calculating and displaying indicator and performance information
- Computing timing variables
- Displaying Z-buffered, lighted, Gouraud-shaded, terrain
- Displaying platforms

By using the hardware to draw very large quantities of polygons, MPS II gives a good indication of the capabilities of the special purpose graphics hardware of the Silicon Graphics Inc. IRIS 4D/70GT in the environment of an applications program. Additionally, MPS II provides an excellent tool to test the performance of the workstations on a wide variety of picture complexities by providing the choice of drawing several different resolutions of terrain.

When performing the evaluation of MPS II, the benchmark test set forth in [Ref. 1:p. 62] was adhered to as closely as possible. Table 9-1 gives the results of that test. If we compare these results to the results of similar evaluations of MPS (see Table 1-4), we see a slight improvement with the performance of MPS II. These

TABLE 9-1 MPS II PERFORMANCE MEASUREMENTS on an IRIS 4D/70GT

DISPLAYING DETAILED TERRAIN

PLATFORM	ZOOM ANGLE (DEGREES)	POLYGONS PER FRAME	FRAMES PER SECOND
ONE VEHICLE	55	764	10.80
ONE VEHICLE	15	404	13.75
NINE VEHICLES	55	1086	7.50
NINE VEHICLES	15	723	9.30
MISSILE 1500m	90	19802	1.02
MISSILE 1500m	10	3388	3.30

DISPLAYING ATTENUATED TERRAIN

PLATFORM	ZOOM ANGLE (DEGREES)	POLYGONS PER FRAME	FRAMES PER SECOND
ONE VEHICLE	55	608	11.50
ONE VEHICLE	15	394	14.00
NINE VEHICLES	55	941	8.67
NINE VEHICLES	15	681	9.80
MISSILE 1500m	90	4152	3.90
MISSILE 1500m	10	816	9.80

improvements can be attributed to using the **mesh** drawing primitive. The performance statistics indicate that, for a relatively small number of polygons (~300), the mesh provides little or no improvement in performance. But, when drawing several thousand polygons, an increase of several frames per second is realized.

Although the comparison between the performance of MPS and MPS II is relevant, it is also important to evaluate the performance of MPS II while drawing higher resolution terrain. Table 9-2 provides the results of those evaluations.

TABLE 9-2 MPS II PERFORMANCE DRAWING HIGH RESOLUTION TERRAIN

DISPLAYING ATTENUATED TERRAIN

RESOLUTION	PLATFORMS	ZOOM ANGLE (DEGREES)	POLYGONS PER FRAME	FRAMES PER SECOND	POLYGONS PER SECOND
100	ONE	55	608	11.5	6992
75	ONE	55	853	9.6	7630
50	ONE	55	2232	5.3	11830
25	ONE	55	8266	1.9	15705
12.5	ONE	55	30914	0.5	15475

DISPLAYING DETAILED TERRAIN

RESOLUTION	PLATFORMS	ZOOM ANGLE (DEGREES)	POLYGONS PER FRAME	FRAMES PER SECOND	POLYGONS PER SECOND
100	ONE	55	763	10.0	7630
75	ONE	55	1310	7.5	9825
50	ONE	55	2834	4.3	12186
25	ONE	55	11700	1.4	16830
12.5	ONE	55	46544	0.4	17687

Figure 9-1 is a comparison graph showing the gap between manufacturer's specification and the achieved performance of MPS and MPS II. The gap between the two applications and the manufacturer's specification is easily understood. The manufacturer's benchmark was designed to produce the maximum possible drawing performance. In this case, the benchmark did little or no computations between frames, whereas MPS and MPS II are computation intensive. Another important factor affecting the display performance is the size of the polygons displayed. The smaller

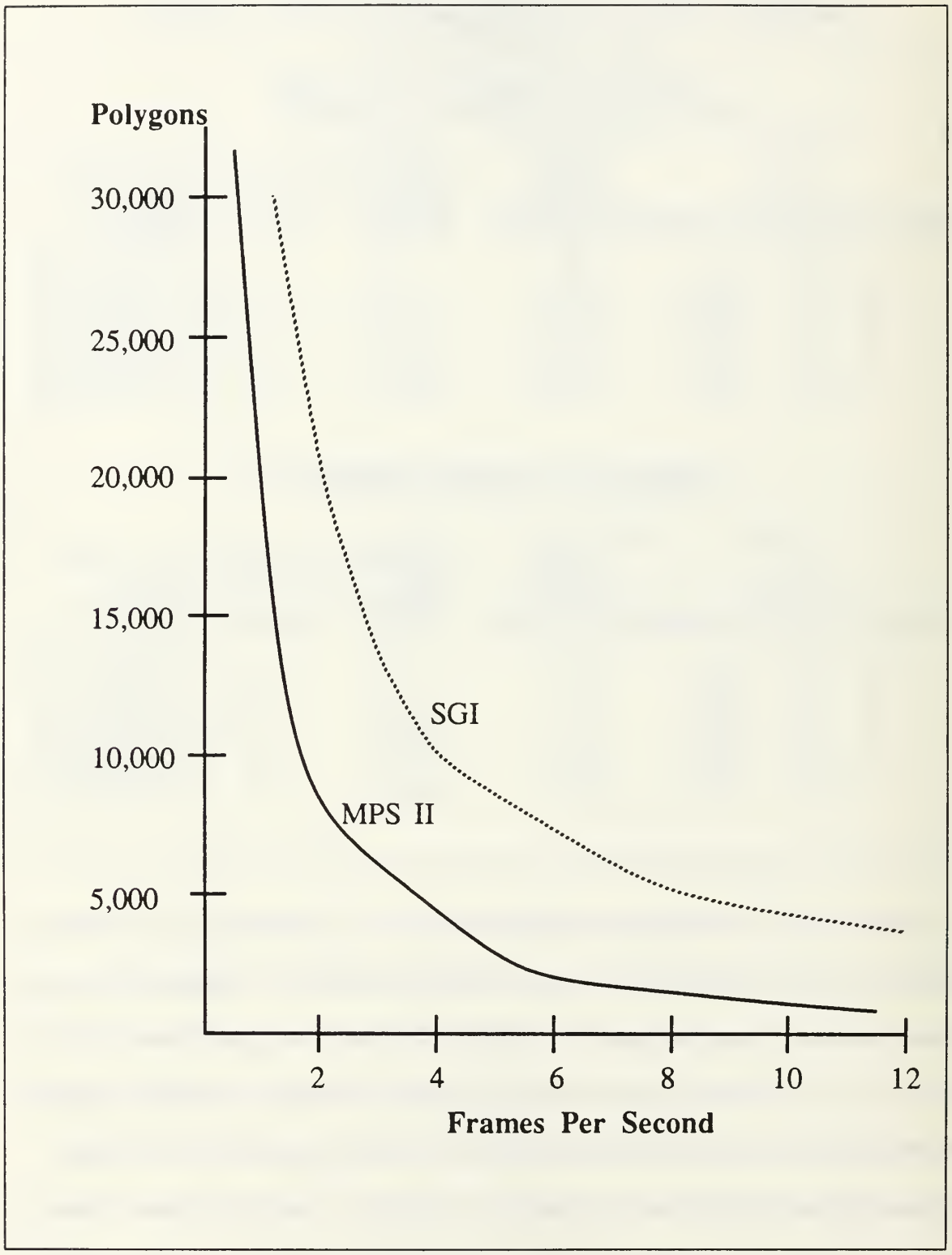


Figure 9-1 Performance Measurement Comparisons

the polygons, the faster they are displayed. When displaying higher resolution terrain, MPS II draws much smaller polygons; therefore, the polygons per second display rate is much higher. It should also be noted that as the frames per second rate decreases, the polygons per second rate increases. This can be attributed to the fact that the percentage of time spent doing computations other than drawing is less.

Table 9-3 gives the performance statistics of MPS II when operating in the network mode, displaying real-time vehicles, and simultaneously saving intervisibility information sent by the LOS module. The performance is not severely degraded even when handling 30 platforms.

B. SYSTEM LIMITATIONS

The performance of MPS II is degraded to unacceptable levels for use as a simulator when displaying terrain of resolution greater than 50 meters. But as can be seen in the photographs in Figure 9-2 and Figure 9-3, the quality of the 12.5 meter resolution is significantly better than 100 meter resolution.

When displaying high resolution terrain, the system's performance falls below one frame per second, and the system controls become unresponsive and inadequate. The user receives visual feedback from the system so infrequently, he has no feel for the sensitivity of the controls. This causes overreaction, which only compounds the problem. Special control software must be written to compensate for these conditions when displaying higher resolution terrain until faster graphics hardware becomes available.

TABLE 9-3 MPS II PERFORMANCE DRAWING HIGH RESOLUTION TERRAIN in the NETWORK MODE

DISPLAYING ATTENUATED TERRAIN IN THE NETWORK MODE

RESOLUTION	PLATFORMS	ZOOM ANGLE (DEGREES)	POLYGONS PER FRAME	FRAMES PER SECOND	POLYGONS PER SECOND
100	11	55	608	8.9	5411
100	30	55	608	8.2	4985
75	11	55	758	8.2	6216
50	11	55	2310	4.3	9933
50	30	55	2104	4.3	9047
25	11	55	8294	1.6	13270
12.5	11	55	30738	0.5	15369

DISPLAYING DETAILED TERRAIN IN THE NETWORK MODE

RESOLUTION	PLATFORMS	ZOOM ANGLE (DEGREES)	POLYGONS PER FRAME	FRAMES PER SECOND	POLYGONS PER SECOND
100	11	55	763	8.3	6332
100	30	55	763	7.3	5570
75	11	55	1271	6.5	8261
50	11	55	2814	4.0	11256
50	30	55	2697	3.7	9978
25	11	55	10905	1.4	15049
12.5	11	55	43214	0.4	16421

It should also be noted that the extremely large data structures used by MPS II affect portions of the program not within the drawing loop, and the effect that the amount of system memory has on its performance. Most noticeable is the response of the pop-up menus. If MPS II is operated on a workstation possessing

Figure 9-2 Terrain Display Using 100 Meter Resolution

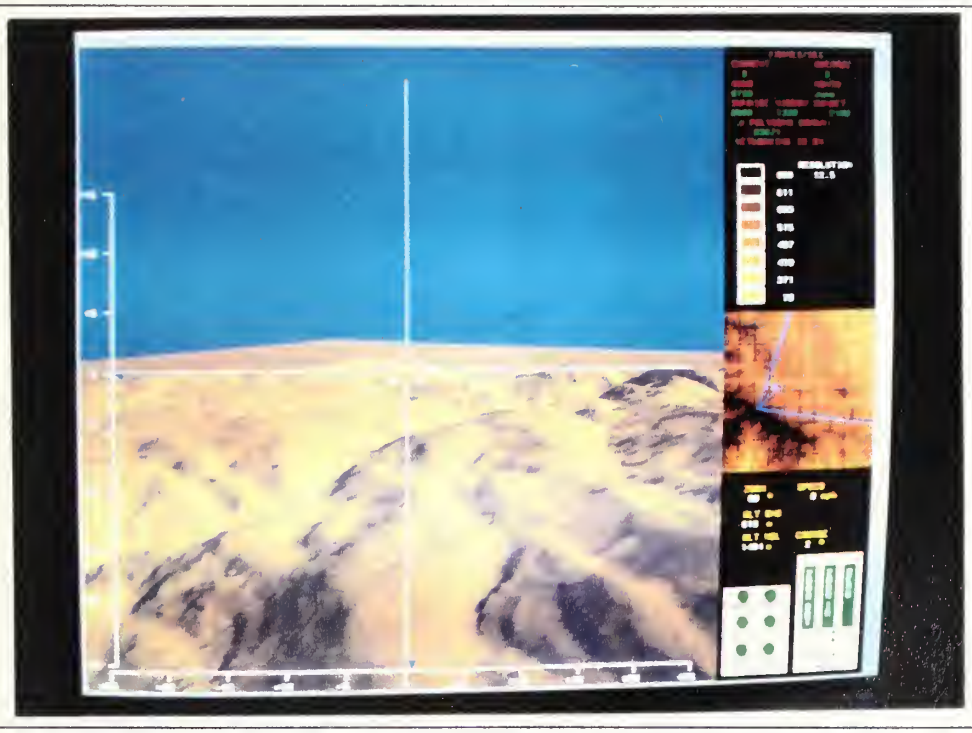


Figure 9-3 Terrain Display Using 12.5 Meter Resolution

eight megabytes of CPU memory, the user must wait approximately five seconds for the menu to appear. To a novice user, this can be confusing. In contrast, when MPS II is run on a workstation with 16 megabytes of memory, the pop-up menu response is almost instantaneous. By using a software evaluation tool called Graphic Operating System View (gr_osview), it was discovered that before a menu could be displayed on the eight-megabyte system, the menu definitions had to be swapped in from disk (virtual memory). This virtual memory operation is much slower than normal memory access, and is the cause of the delay.

The memory limitation is also a problem when trying to display 12.5 meter resolution terrain on the eight megabyte system. When the terrain data structures are being built, a thrashing problem arises. The processor spends so much of its time swapping data in and out of virtual memory, little of the needed terrain computations are ever accomplished. The results are that the preprocessing routines require a very long time to build the terrain and vertex normals data structure (approximately 30 minutes). In contrast, the 16 megabyte system only takes approximately five minutes to complete the same computations.

X. CONCLUSIONS AND FUTURE WORK

A. CONCLUSIONS

This work presents enhancements to the Moving Platform Simulator. Designated the Moving Platform Simulator II, this new version of the Moving Platform Simulator was designed to fulfill some specific needs of USACDEC and, at the same time, provide a vehicle with which to pursue research into the design, implementation, and performance analysis of moving platform simulators. The successes that were achieved through this research include:

- Improved drawing algorithms that provide more realistic looking terrain
- Display of higher resolution terrain
- Integration of real-time, actual platform data into the simulator to control the position and movements of displayed platforms
- Presentation of a distributed computing architecture to maximize display performance
- Intervisibility determinations and display
- Increased simulator performance

By providing a display of real-time platforms over more realistic terrain, MPS II has made large strides towards meeting USACDEC's requirement for a comprehensive moving platform simulator. Furthermore, with the platform intervisibility module of MPS II, USACDEC now has a valuable tool to assist them in the evaluation of the FOGM missile, as well as many other systems.

In addition to supporting the needs of CDEC, MPS II provides an excellent vehicle with which to evaluate the performance of the Silicon Graphics IRIS 4D/70GT and other workstations in an application program environment. The ability to

dynamically select for display multiple resolutions of terrain using multiple terrain drawing algorithms allows the user to evaluate the performance of the workstation at many different levels without leaving the program.

An evaluation of MPS II shows improved performance over its predecessor, MPS. The program provides adequate performance as a simulator when displaying up to 50 meter resolution. When displaying higher than 50 meter resolution, MPS II presents an excellent view of the terrain, bit the system performance is not sufficient enough to use MPS II as a moving platform simulator.

B. FUTURE WORK

MPS II evolved from the products of more than two years of student effort. Because the final product was not envisioned from the beginning, it has been designed by several totally separate teams. For this reason, and because the current size of the project (over 30,000 lines of code) MPS II has become difficult to manage.

A logical next step in the evolution of moving platform simulators here at the Naval Postgraduate school is to study all present versions of MPS and its derivatives. The goal of this study would be to produce a list of the best attributes and functionality of each of the simulators and a list of the desired properties of the next generation of the simulator. This list should then serve as the requirements document and as a basis for a comprehensive design project, using software engineering techniques, of the next generation of simulators. Each desired property or functionality should be modularized to allow the simulator to be tailored to meet different operational requirements. This would greatly aid in the production of future simulators and greatly simplify future research.

Because of limited processing power, MPS II uses a system of distributed processes. In fact, to operate the simulator with all options and capabilities available a distributed computing architecture is used and four different processes, possibly on four different machines, must be running. With the advent of the more powerful graphics workstations, consolidating MPS II to run entirely on one workstation should be investigated.

The next evolution of the terrain display should be to incorporate the display of cultural features into the display. This data, which is included in the DMA DFAD data file, includes vegetation and man made objects.

Finally, to improve the accuracy of the intervisibility determinations, the effects of weather, vegetation, and man made obstructions should be included into the calculations.

APPENDIX A USER INTERFACE

The user interface of any application program must be designed so that novice and experienced users alike can effectively operate the program with little or no help from user's manuals or other users. A thorough and efficient design of command line options, pop-up menus, dials, and a mouse achieves this.

A. MPS II USER INTERFACE

The user interface of MPS II is, for the most part, identical to that of the original MPS. Most changes are merely additions or deletions to pop-up menus. For this reason, much of this chapter is a reproduction of the User Interface manual for MPS, with the appropriate changes made to explain the new features available in MPS II [Ref. 1:pp. 74-88]. There are also new sections covering the execution of the Network Simulator and the Line-of-Sight (LOS) Module. For instructions on the user interface of the Line-of-Sight Module, see Section B of this Appendix.

1. STARTING THE SIMULATOR (COMMAND LINE OPTIONS)

To initially start MPS II, while at the UNIX prompt and in the directory containing the program, type the command **mps [-ntslb]**. The optional command line options are described below.

2. COMMAND LINE OPTIONS

MPS II currently has five options available from the command line[7].

- Network mode
- Test mode
- Silent mode
- Start Line-of-Sight (LOS) module remotely in the foreground
- Start Line-of-Sight (LOS) module remotely in the background

Selection of **-n** (network mode) activates the networking capabilities of the program. If one or more MPS processes, the network simulator, and/or the LOS module are operating on different machines, they will be able to share information regarding the other platforms. When a platform changes course, speed, or altitude (FOGM only), a broadcast packet is sent to all other processes and the appropriate platform's information is updated. This mode must be selected to be able to start the Network Simulator or to remotely run the LOS module.

Selection of the **-t** (test mode) option bypasses some of the cosmetic portions of the program. Currently, the only part that is bypassed is the opening billboard sequence.

Selection of the **-s** (silent mode) option turns off the bell that rings to indicate acceptance of input from the user. This option is useful for demonstrations when the ringing would interfere with a verbal explanation of the program.

---

[7]The code that processes the command line arguments is contained in the file *decode_arguments.c*.

If the **-l** (LOS foreground mode) option is used, the LOS module is automatically started on a remote processor[8]. The LOS module will be run in the foreground[9] on that machine so that line-of-sight information will be displayed on the remote IRIS's screen, and user input will be accepted on the remote IRIS in this mode (see Section B of this Appendix for further details). The -n (network mode) must also be selected for the -l option to work.

Selecting the **-b** (LOS background) option automatically starts the LOS module in the background on a remote IRIS. Since the LOS module process is running in the background on the remote IRIS, no display is seen on the remote machine, and no user inputs are accepted from that processor. All controls to the LOS module must be given remotely using the pop-up menus on MPS II described later in this section. As with the -l option, the -n option must be selected in conjunction with this option.

3. POP-UP MENU SYSTEM

Pop-up menus are the primary source of user input into the program. There are currently 27 different pop-up menus that are used in various parts of the simulation. If a selection in a menu is not allowed or is meaningless when the menu is displayed, the selection is displayed in lower case. Otherwise, the selection is completely uppercase. Disallowed selections were not omitted so that the menus

---

[8]The remote processor on which the Line-of-Sight module is executed on is defined in the file *files.h*.

[9]A user must be logged onto the remote IRIS and the window manager must be running before the remote foreground start is attempted.

appear in the same order and format every time. If disallowed selections were eliminated, users would tend to be overwhelmed by the number of different menus. In fact, of the 24 menus in the system, only 13 are really unique. Some of the menus are only reachable as roll-offs of other menus.

After the MPS II billboard is displayed, the MPS II introduction screen is displayed. The initial database read is immediately begun, which is indicated by a moving wait bar. Upon completion of reading the terrain database, the 35 kilometer map is displayed, as well as a message to press the right mouse button for a pop-up menu.

Since MPS II must display real-time vehicles, pop-up menus do not automatically appear. Pop-up menus temporarily halt program execution while waiting for user selection. Instead, moving icons of any real-time or simulator platforms are displayed at this point in the execution of the program. When the right mouse button is held down, the first pop-up menu appears[10]. A detailed explanation of each menu follows:

a. Select Area Menus

The pop-up menu display at the 35 kilometer map is titled SELECT_AREA_MENU. This menu contains the following ten selections:

- CONTINUE
- SELECT AN AREA OF THE MAP
- GO TO MAIN MENU
- START NETWORK SIMULATOR

[10]Users should avoid staying in pop-up menus for extended periods of time when running MPS II in the network mode. Long delays can cause the input buffer to overflow, causing the loss of some incoming information from the network.

- KILL NETWORK SIMULATOR
- CHOOSE DATA RESOLUTION
- EXIT THE PROGRAM
- ENTER 4SIGHT (RESIZE OPTIONS)
- COLOR SCHEME
- LOS MENU

By selecting **CONTINUE**, the user continues the simulation with no changes. The second option allows the user to select a ten kilometer area of operation.

Selecting **GO TO MAIN MENU** will take the user to the main menu, which is the next logical place to go after selecting an area in which to operate.

When chosen, the **START NETWORK SIMULATOR** option remotely starts the FHL Network Simulator, which presently runs on a VAX 11/785. After approximately one minute, real-time vehicles will begin being displayed on the 35 kilometer map. The status window on the right-hand side of the screen displays the current status of the Network Simulator. If the Network Simulator is already running, no attempt should be made to start it. If the user attempts to start the Network Simulator when not in the network mode, a warning will be displayed.

Selection **KILL NETWORK SIMULATOR** will send a message to the Network Simulator instructing it to die. Vehicles presently being displayed as network vehicles will continue to be displayed with their last course and velocity.

The next menu option is a roll-off menu selection. By highlighting the option and then moving the mouse to the right, the **RESOLUTION_MENU** will appear. The user then may select from a choice of possible terrain data resolutions ranging from 12.5 to 100 meters.

To exit the program, the user must select **EXIT THE PROGRAM** and a small menu will be displayed with the following selections:

- RETURN TO WHERE YOU WERE
- REALLY QUIT

If the user desires to resize or move the simulation's windows, the option **ENTER 4SIGHT (RESIZE OPTIONS)** will allow him to accomplish this. After selecting this option, the windows will be cleared to white, and the user can click on the menu bar and move or resize as desired.

The next option, **COLOR SCHEME**, is another roll of menu with the following options:

- COLOR SCHEME - BROWN RAMP
- COLOR SCHEME - MULTIPLE COLORS
- COLOR SCHEME - GREY RAMP
- COLOR SCHEME - RED RAMP
- COLOR SCHEME - GREEN RAMP
- COLOR SCHEME - BLUE RAMP

The color scheme selections change the way the terrain is colored. Each color scheme has 16 different colors that are based on the elevation at that location.

The next selection, **LOS MENU**, is another roll-off menu with the following options:

- CONTINUE
- POSITION OBSERVER
- SELECT VEH TO DRAW LOS
- DESELECT VEH TO DRAW LOS
- MAP MENU
- START LOS MODULE
- KILL LOS MODULE
- EXIT THE PROGRAM

This menu is for remote control of the LOS module. These options are a subset of the options found when running the LOS module in the foreground on another IRIS. See Section B of this Appendix for a detailed description of these options.

b. Main Menus

There are four menus that make up the main menu set. These menus are called MAIN_ONE, MAIN_TWO, MAIN_THREE, and MAIN_FOUR. Each of these menus contains the same eight selections as follows:

- PLACE DEFAULT SET OF PLATFORMS
- ADD A PLATFORM
- DELETE A PLATFORM
- SELECT A PLATFORM TO OPERATE
- EXIT THE PROGRAM
- ENTER 4SIGHT (RESIZE OPTIONS)
- SAVE PLATFORMS TO A FILE
- RETURN TO 35 KM MAP
- CHOOSE DATA RESOLUTION

MAIN_ONE is the first menu that is displayed after selecting an area of the map. Since there are no platforms displayed at this point, the delete, select, and save options are disallowed. After adding at least one platform, MAIN_TWO is displayed, which allows all selections on the menu. MAIN_THREE is displayed only when the act of adding default sets of platforms would exceed an arbitrary limit on the number of platforms allowed in the simulation at any one time. MAIN_FOUR is displayed when the limit on the number of platforms displayed has been reached.

Selecting the first option (**PLACE DEFAULT SET OF PLATFORMS**) will display another menu called DEFAULT_MENU. This menu contains 6 selections as follows:

- ENTER THE FILENAME FOR YOUR PLATFORMS
- CONVOY - 10 GROUND PLATFORMS
- CONVOY - 10 GROUND & 1 FOGM PLATFORM
- JEEPS - 20 IN A ROW
- DR. ZYDA'S CONVOY
- DR. ZYDA'S WILDMAN DEFAULTS

If the user selects the first option, a small window is displayed on the screen which prompts the user for the filename. If valid information is found in the file, the appropriate platforms are added to the simulation. The main menu is then redisplayed.

Selection of any other option on the **DEFAULT_MENU** results in the addition of predesignated platforms in predesignated locations. These selections are useful for demonstration purposes and for persons interested in getting some platforms on the screen very quickly.

The information for the default sets of platforms is contained in data files that are read when indicated by a menu selection. The complete path for these files is contained in the header file files.h.

The next option on the main menu is **ADD A PLATFORM**. Selecting this option displays the following menu:

- ADD A COVERED JEEP
- ADD AN OPEN JEEP
- ADD A TRUCK
- ADD A TANK
- ADD A FOGM MISSILE
- ADD AN OBSTACLE

If a moving platform is selected (jeep, tank, truck, or FOGM), menus are displayed requesting an initial speed and direction for the platform. If an obstacle is requested, then the speed and direction menus are bypassed. The FOGM missile defaults to an initial altitude of 200 meters above the terrain at the point where it is placed. After completing the selections, an icon is placed on the screen that resembles the selected platform or obstacle. The user can then move the icon with the mouse and place the platform by clicking the right mouse button. After placing the icon on the screen, the main menu is displayed once again.

Selecting the **DELETE A PLATFORM** option displays the following menu:

- DELETE A SINGLE PLATFORM
- DELETE ALL PLATFORMS ON THE SCREEN

If the user wants to delete one platform, an X cursor is displayed and the user can click on the desired platform. If the user wants to delete all the platforms on the screen, the following menu is displayed:

- NO, DO NOT DELETE ALL THE PLATFORMS
- YES, DELETE ALL PLATFORMS

The appropriate selection from this menu either cancels the operation or executes it. This menu prevents a user from deleting vehicles that he may not really want to delete.

The next selection from the main menu is **SELECT A PLATFORM TO OPERATE**. If the user selects this option, the following menu is displayed:

- ZOOM IN TO ANY LEGAL GRID SQUARE
- SELECT A PLATFORM TO OPERATE RIGHT NOW

The zoom option is usually necessary if platforms are close to each other, and the individual icons overlap. By zooming into the 1x1 kilometer grid square, the user can more easily select the platform he desires.

If the platform the user wants to operate is clearly visible, then the second selection allows the user to select a platform immediately.

If the user has placed platforms on the screen and wishes to save them to a file, then the main menu selection **SAVE PLATFORMS TO A FILE** accomplishes this. A window opens that prompts the user for the filename. If the path is correct, the platforms are saved to the file.

The last selection from the main menu allows a user to return to the **SELECT_AREA** menu.

c. Operating Menus

(1) Driving. There is only one menu that makes up the driving menu set. This menu is called OPERATE_DRIVE. This menu contains the seven selections as follows:

- DO NOTHING
- RETURN TO MAIN MENU
- CHANGE ALL PLATFORMS' SPEEDS
- EXIT THE PROGRAM
- ENTER 4SIGHT (RESIZE OPTIONS)
- ADVANCED OPTIONS
- CHOOSE DATA RESOLUTION

The first selection is provided in case the user pushes the right mouse button and he does not desire to do anything. The second selection allows the user to return to the main menu.

The third selection causes another menu to pop up that allows the user to select a speed for all the platforms currently in the simulation. The allowable speeds are from zero to 65 miles per hour. There is also a selection that will do nothing and return directly to the simulation. Changing all the speeds is convenient when the user wants to have a convoy of platforms proceed at identical speeds. Also, by selecting zero miles per hour, all platforms are effectively frozen and their configuration can be studied by viewing them from a FOGM missile or other platform.

The **ADVANCED OPTIONS** selection brings up the following menu:

- TOGGLE SINGLE/DOUBLE BUFFER MODE
- TARGETING MODE TEST (ONCE)
- TERRAIN DRAWING OPTIONS

The first selection toggles the graphics hardware between single buffer and double buffer modes. In double buffer mode, all drawing is done in a separate area of memory from the display memory. When the function *swapbuffers*() is called, the pointer to this area and the pointer to the display buffer are switched, thereby swapping the new picture for the old picture. This is how smooth motion is simulated. If a user is interested in what order the individual picture elements are drawn on the screen, then by selecting single buffer mode, he can see the pictures while they are being drawn.

TARGETING MODE test allows a user to see how the simulation determines if a target is in the crosshairs of the FOGM missile during targeting. After selecting the option, the next time targeting is attempted, the view will be cleared to white, and all visible platforms will be drawn without lighting, shading,

or hidden surface removal. The resulting picture is displayed for three seconds, and then normal operation commences. This option is reset each time it is used.

The **TERRAIN DRAWING OPTIONS** option is a roll-off menu. When the user moves the cursor towards the right side of the words **TERRAIN DRAWING OPTIONS**, the following menu is displayed:

- DETAILED TERRAIN
- DISTANCE ATTENUATION - NORMAL
- NORMAL CALCULATION METHODS

The default terrain drawing option is **DISTANCE ATTENUATION - NORMAL**. This drawing option establishes three zones in front of the driven platform and reduces the number of polygons that are displayed in each zone. The zone closest to the viewer is displayed with resolution selected. The next zone uses one-half the selected resolution, and the last zone uses one-forth the selected resolution. The selection for **DETAILED TERRAIN** draws full resolution polygons throughout the three zones. Users notice a significant decrease in the frames per second rate when this option is selected. If single buffer mode is also enabled during detailed terrain drawing, the algorithm that is used to draw the terrain becomes more obvious.

The **NORMAL CALCULATION METHOD** selection is a roll-off menu with the following options:

- USE NORMAL APPROXIMATION METHOD
- USE VERTEX NORMAL METHOD

These options let the user select between the two methods of computing vertex normals discussed in Chapter V. The first option uses the normal of the triangle to the

Northeast of the vertex as the normal of the vertex. The second option use the cross product method described.

(2) Flying. There are three menus that make up the flying menu set. These menus are called OPERATE_FLY_ONE, OPERATE_FLY_TWO, and OPERATE_FLY_THREE. This menu contains the seven selections as follows:

- DO NOTHING
- DETACH/RESUME OPERATING
- RETURN TO MAIN MENU
- CHANGE ALL PLATFORMS' SPEEDS
- EXIT THE PROGRAM
- ENTER 4SIGHT (RESIZE OPTIONS)
- TOGGLE TARGET TRACKING
- ADVANCED OPTIONS

Many of these options are exact duplicates of the options on the driving menus. However, the **DETACH/RESUME OPERATING** and **TOGGLE TARGET TRACKING** options are different.

The **DETACH/RESUME OPERATING** option allows a user to detach the cursor from the simulation while flying. During flying, the cursor is restricted to the simulation window because the mouse controls where the nose camera of the FOGM missile is pointed. Using this option, the user can point the camera where he wants to look and then free the mouse. To return to the simulation, the user must select the same option once again.

If the user has a ground platform in the crosshairs of the FOGM missile and he wants to target it, he must make the **TOGGLE TARGET TRACKING** selection from the menu. If a platform was in the crosshairs, then the missile will

lock on and track the platform. If the user wants to release the missile from tracking mode, then another selection will turn off target tracking.

3. DIALS

The dial box that is supplied by SGI has eight dials numbered from zero to seven. They are organized in two columns and four rows. The numbering scheme is from left to right, bottom to top, so the lower left dial is zero, the lower right is one and the upper right is seven.

The Moving Platform Simulator uses these dials in basically two configurations: one for driving and one for flying.

a. Driving Dial Configuration

The dials for driving are configured as follows:

- DIAL 0 - Course
- DIAL 1 - Viewing Direction
- DIAL 2 - Speed
- DIAL 3 - Tilt
- DIAL 4 - Hour of the Day
- DIAL 5 - Month of the Year
- DIAL 6 - Not Used
- DIAL 7 - Not Used

The course is the direction of travel of the platform, which is displayed in degrees[11]. The viewing direction is the direction the driver's head is looking left to right in relation to the course. When the course is changed, the viewing angle changes accordingly. Speed is the speed of the platform in miles per hour. Tilt is where the

[11]Note that the course and speed of a real-time network platform cannot be changed. The view direction and tilt are adjustable, but the operator has no control over the course or speed of a remotely displayed vehicle.

driver is looking up and down. The hour of the day and month of the year determine the location, color, and intensity of the sun. Figure A-1 is a picture of the dial box with the dials labeled for driving.

b. Flying Dial Configuration

The dials for flying are configured as follows:

- DIAL 0 - Course
- DIAL 1 - Altitude
- DIAL 2 - Speed
- DIAL 3 - Not Used
- DIAL 4 - Hour of the Day
- DIAL 5 - Month of the Year
- DIAL 6 - Not Used
- DIAL 7 - Not Used

Many of the dials are identical to the driving dial configuration except for altitude which is self-explanatory. Figure A-2 is a picture of the dial box with the dials labeled for flying.

4. MOUSE

The mouse has many uses throughout the simulation. Its use can be broken down into basically four groups:

- Pop-up menu activation and selection
- Operating area selection
- Platform icon placement and selection
- FOGM missile nose camera control

The mouse is used throughout the simulation to activate pop-up menus and to select options. One of these options is to select an area from the large database. A 10x10 kilometer red square is displayed on the 35x35 kilometer database, and the

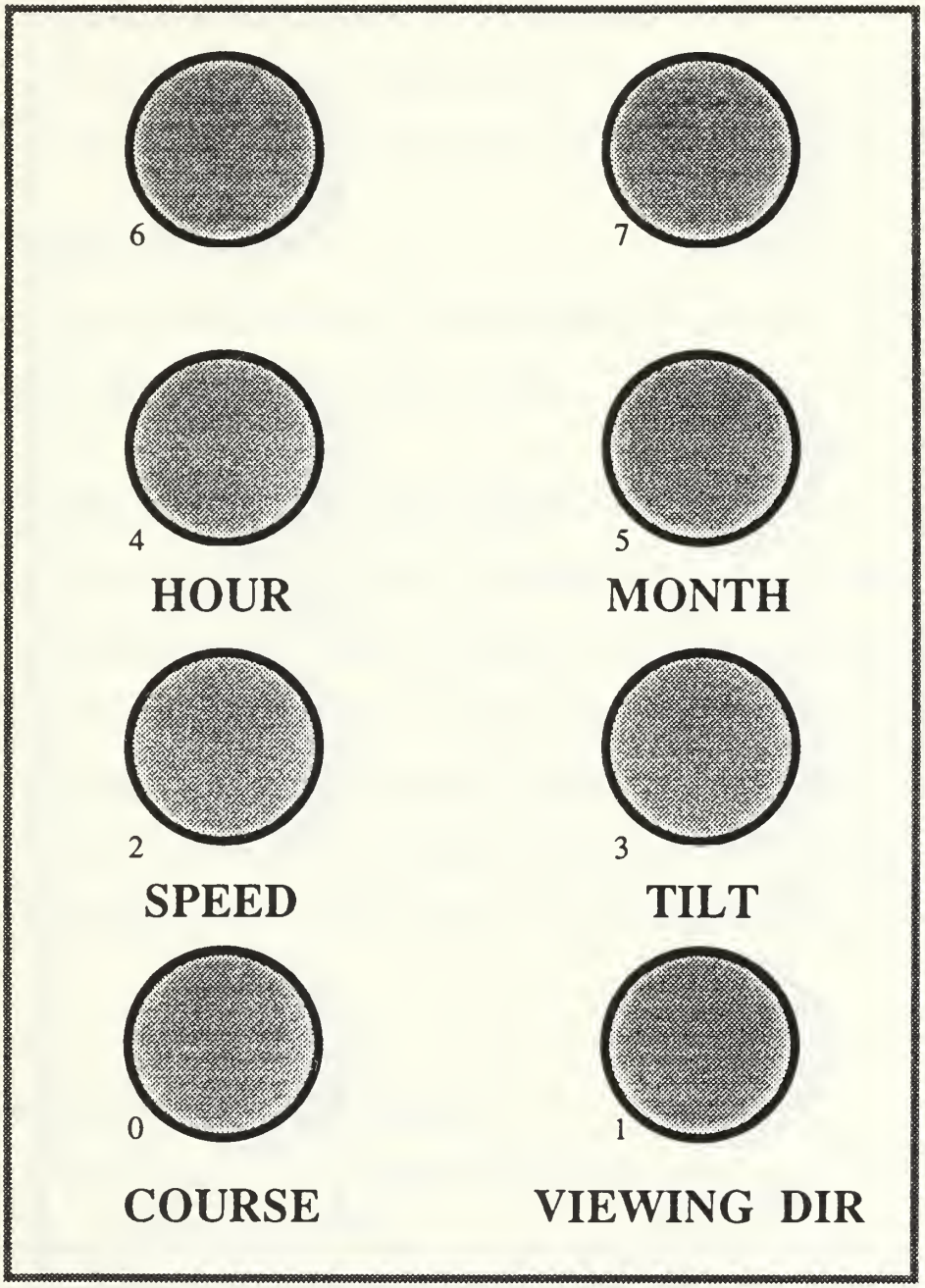


Figure A-1 Dial Box With Dials Labeled For Driving

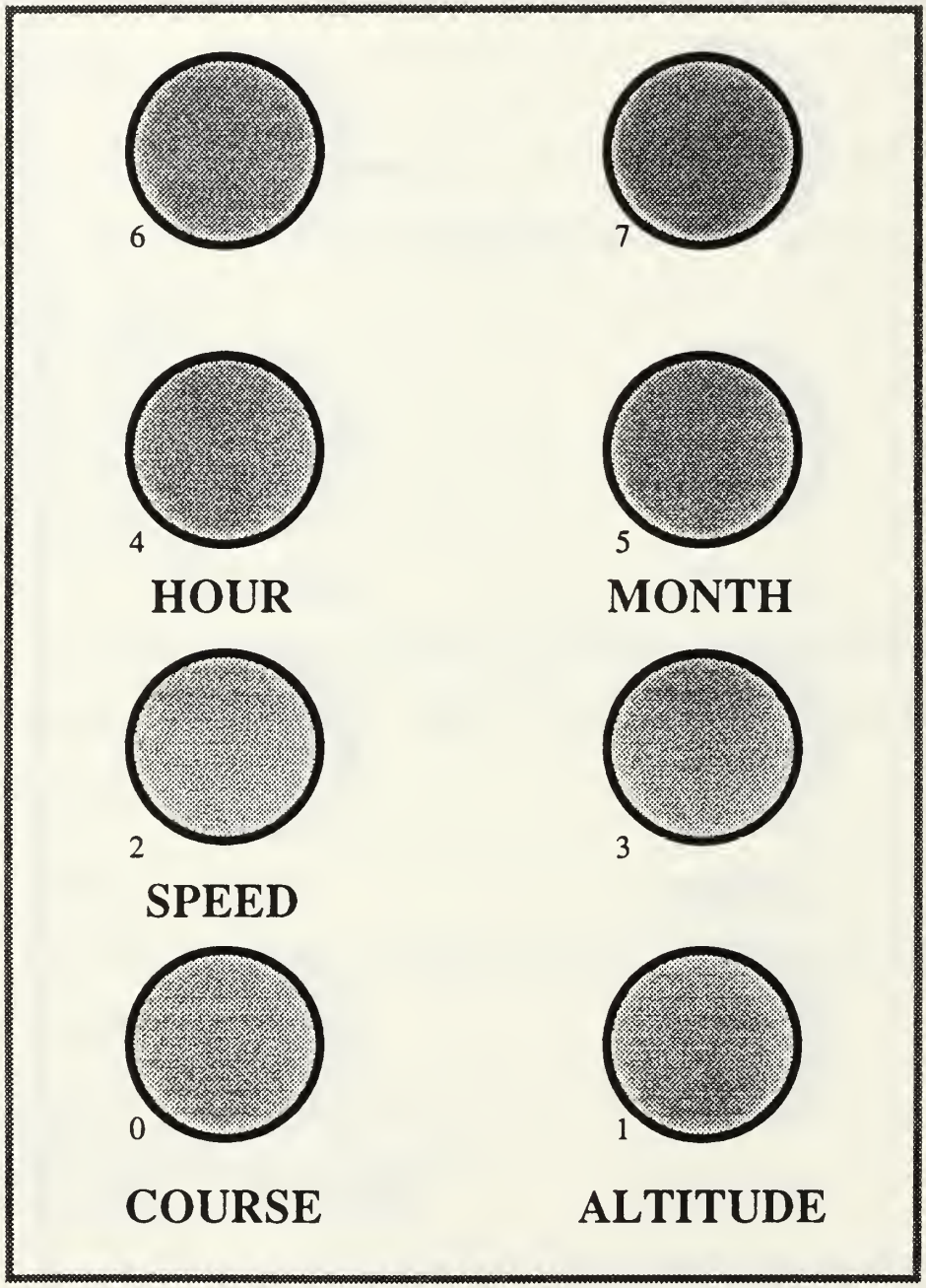


Figure A-2 Dial Box With Dials Labeled For Flying

mouse is used to move the square to the desired location. Platforms are placed and selected on the screen with the mouse.

The nose camera of the FOGM missile is controlled with the movement of the mouse. This gives the user very fine control over targeting and viewing direction.

5. KEYBOARD

The keyboard is only used to accept filenames from the user. All other user input is through the pop-up menus, dials, or mouse.

B. NETWORK_SIMULATOR USER INTERFACE

The user interface of the NETWORK_SIMULATOR module is limited to one command line argument. This argument specifies the data file containing the VIDS data blocks to be used by the module to load the LAN. The module has neither a graphics display nor, except for error messages, output to the terminal screen or printer. All output is to the LAN. To start the module, while at the UNIX prompt, type the command **NETWORK_SIMULATOR** *filename*, where *filename* is the name of the data file.

C. PROCESS_VDB USER INTERFACE

Like the user interface of the NETWORK_SIMULATOR module, the interface of the PROCESS_VDB module is limited to one command line argument. In this case, the argument specifies the number of platforms for which the module will broadcast update messages. The module also has neither a graphics display nor, except for error messages, output to the terminal screen to printer. All output is to the LAN.

To start the module, while at the UNIX prompt, type the command **PROCESS_VDB** *X*, where *X* is the desired number of platforms. If *X* is not specified or is declared to be zero, a default list of vehicles is used. In the file "process_vdb.h", a mechanism exists to select the default list. By defining the manifest constant "HANDLE_VEHICLE_X", where X is the platforms player position message identification number, the platform will be selected.

Since the PROCESS_VDB module functions as the client and the NETWORK_SIMULATOR module functions as the server in the client-server model of interprocess communication used between the two modules, the NETWORK_SIMULATOR module must be running when the PROCESS_VDB module is started.

D. PROCESS_LOS USER INTERFACE

The user interface of the PROCESS_LOS module consists of command line options and pop-up menus activated by pressing the right mouse button.

1. Starting The PROCESS_LOS Module

As discussed in Chapter VIII, the PROCESS_LOS module runs in one of two modes. Selecting the desired mode is done with a single command line argument. If the module is started with the argument *L*, the module will run with displays. Otherwise, the module will run without any graphical display. The *L* stands for "local displays," vice using the remote display capabilities of the MPS II module to display the intervisibility data. To start the PROCESS_LOS module, while at the UNIX prompt, type the command **PROCESS_LOS [L]**.

2. Pop-up Menu System

Control of the PROCESS_LOS module after it is started in the "local display" mode is through pop-up menus. The module uses one main menu with three roll-off menus. When the right mouse button is held down, the main menu appears. The button may have to be held down for a few seconds before the menu appears because the module will finish processing the update message it is currently working on before handling the request for the menu. Additionally, the menu will continue to reappear after each selection until the *Continue* selection is made. Since many times the user will need to make multiple selections from the menu, this speeds up the selection process. However, care must be taken not to remain in the menus for extended periods of time. As long as the user is in the menu, the module is not processing platform update messages, although the module is continuously receiving messages through the network. If the user stays in the menu too long, the network buffers will overflow and messages will be lost. A detailed explanation of each menu follows.

a. Main Menu

The primary menu that first appears whenever the right mouse button is pressed is the *Main Menu*. This menu contains the following 13 selections:

- Continue
- Position Observer
- Place Observer in Vehicle
- Select Vehicles
- Deselect Vehicles
- Select a 1km x 1km Area
- Select a 10km x 10km Area
- Set LOS Segment Length

- Set VEH Segment Length
- Set Map Color Scheme
- Map Menu
- Start Simulator
- EXIT THE PROGRAM

By selecting **Continue**, the user leaves the menu and the module proceeds to process platform update messages it receives.

The **Place Observer in Vehicle** selection allows the user to designate one of the available platforms as the observer platform, and colocate the observer at that platform's location. The observer's position will move with the platform's position. Upon making the selection, a second menu appears. This menu, titled the **Place Observer in Veh Menu,** lists all the available platforms and allows the user to select the desired platform. After the selection is made, the main menu selection will change to **Remove Observer from Vehicle**. Making this selection will detach the position of the observer from that of the platform. The observer will remain at its current location when the selection is made.

The **Select Vehicles** option allows the user to choose the platforms to display. Upon selecting this option, a different menu listing the available platforms, that are not already selected, appears. This menu, titled the **Select Vehicle Menu**, also allows the user to choose all the platforms with one selection by choosing the **Select All Vehicles** option. Even though a platform is not selected for display, the module still determines its intervisibility with the observer. The information is stored for display at a later time.

The **Deselect Vehicles** option allows the user to remove platforms from the list of vehicles to display. Upon selecting this option, a different menu

listing the currently selected platforms appears. This menu, titled the **Deselect Vehicle Menu**, has two other options:

- Deselect All Vehicles
- Remove All Vehicles

The first allows the user to deselect all the platforms with one selection. The second option deselects all the platforms and clears the files containing the intervisibility data.

The **Select a 1km x 1km Area** option allows the user to focus on a 1km x 1km map area. Upon making this selection, a box outlining a 1 km x 1km area in the center of the map display appears. By moving the mouse, this box can be moved to outline a different area. When the right mouse button is pressed again, the map display will change to show the area outlined by the box. Once this option is selected, the option disappears from the main menu and is replace by **Return to 35km x 35km Area.**

The **Select a 10km x 10km Area** functions similarly to the **Select a 1km x 1km Area**. However, in this case, the box outlines a 10km x 10km area and the map display changes to display this area. Additionally, once a 10km x 10km area has been selected, when the user is displaying a 1km x 1km map area, the selection **Return to 10km x 10km Area** appears. The option to select a 10km x 10km area is not available when the user is in the 1km x 1km area.

The **Set LOS Segment Length** option allows the user to adjust the length of the *los_seg_length* (see Chapter VIII). The option is a roll-off menu to the **LOS Segment Length Menu**. By highlighting the option and then moving the mouse to the right, a second menu will appear. This second menu has the following options:

- Segment Length = 5 meters
- Segment Length = 10 meters
- Segment Length = 20 meters
- Segment Length = 50 meters
- Segment Length = 100 meters
- Segment Length = 150 meters
- Segment Length = 200 meters
- Segment Length = 300 meters

The length of the *los_seg_length* can be changed by selecting the desired length from the menu.

The **Set VEH Segment Length** option is also a roll-off menu and functions similarly to the **Set LOS Segment Length** option. However, in this case the *veh_seg_length* is adjusted. The roll-off menu is titled the **VEH Segment Length Menu**.

The next option, **Set Map Color Scheme**, is another roll-off menu with the following options:

- COLOR SCHEME - BROWN RAMP
- COLOR SCHEME - MULTIPLE COLORS
- COLOR SCHEME - GREY RAMP
- COLOR SCHEME - RED RAMP
- COLOR SCHEME - GREEN RAMP
- COLOR SCHEME - BLUE RAMP

The color scheme selections change the way the terrain is colored. Each color scheme has 16 different colors that are based on the elevation at that location.

The last roll-off menu is the **Map Menu**. This menu controls the display of the intervisibility data. The menu rolls off to another menu also called the **Map Menu**. The options available from this menu are listed below. For those options

that toggle from one option to another when selected, the initial default option is listed first.

- Clear Map
- Redraw w/LOS Segments Only
- Redraw w/VEH Segments Only
- Redraw w/VEH and LOS Segments
- Draw LOS Trace / DON'T Draw LOS Trace
- DON'T Draw VEH Trace / Draw VEH Trace
- DON'T Draw VEH / IDs Draw VEH IDs
- Draw VEH Path / Draw VEH Positions

The first four options clear the map of all intervisibility drawings. The **Redraw w/LOS Segments Only, Redraw w/VEH Segments Only,** and **Redraw w/VEH and LOS Segments** options then will draw the line-of-sight path from the observer to the selected target platforms, the path of the target platforms, or both, respectively. The **Draw LOS Trace** selection toggles the drawing mode to display the display the line-of-sight paths as they are determined. This option will toggle to **DON'T Draw LOS Trace** if it is selected. Similarly, the **DON'T Draw VEH Trace** option disables the drawing of the platform path. This option will toggle to **Draw VEH Trace** if it is selected. The **DON'T Draw VEH IDs** option disables the drawing of the platform identification numbers adjacent to the platform's path. When selected, the option changes to **Draw VEH IDs**. The final option on this roll-off menu is **Draw VEH Path**. When selected, the drawing of the platform's path changes from drawing only the actual point where the intervisibility was determined (the target point) to drawing the entire path of the vehicle. The selection will toggle to **Draw VEH Positions** if it is selected.

The second to the last option on the main memu is **Start Simulator**. When chosen, the option remotely starts the NETWORK_SIMULATOR and PROCESS_VDB modules. These processes are presently set to run on the VAX 11/785. After approximately one minute, real-time vehicles will begin to appear. The current status of the processes is displayed in the statistics window.

The last option on the main menu is **EXIT THE PROGRAM**. When selected, the module will terminate.

LIST OF REFERENCES

1. Fichten, Mark A., and Jennings, David H., *Meaningful Real-Time Graphics Workstation Performance Measurements,* M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1988.

2. Smith, Douglas B., and Streyle, Dale G., *An Inexpensive Real-Time Interactive Three-Dimensional Flight Simulation System,* M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1987.

3. Oliver, Michael R., and Stahl, David J., *Interactive, Networked, Moving Platform Simulators,* M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1987.

4. Silicon Graphics Inc., *IRIS User's Guide, MEX Window Manager,* Mountain View, California, 1987.

5. Silicon Graphics Inc., *4Sight User's Guide,* v. 1, Mountain View, California, 1988.

6. Defense Mapping Agency, *Product Specifications for Digital Landmass System (DMLS) Data Base,* 2d ed., April 1983.

7. Drummond, William T., Nizolak, Joseph P., *A Graphics Workstation Field Artillery Forward Observer Simulator Trainer*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1989.

8. Hearn, Donald and Baker, M. Pauline, *Computer Graphics*, pp. 262-264, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1986.

9. Silicon Graphics Inc., *IRIS User's Guide*, v. 1, Mountain View, California, 1988.

10. Shannon, Larry R., Teter, William A., *The Autonomous Vehicle Simulator*, M.S. Thesis, Naval Postgraduate School, Monterey California, June 1989.

11. Hyzer, William G., *Engineering and Scientific High-Speed Photography*, The MacMillian Company, New York, 1962.

12. Barrow, Theodore H., *Distributed Computer Communications in Support of Real-Time Visual Simulations, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1988.*

13. Silicon Graphics, Inc., *Power Series A Family Overview,* Mountain View, California, September 1988.

14. Stallings, William, *Data and Computer Communications*, MacMillan Publishing Company, New York, 1988.

15. Comer, Douglas E., *Internetworking with TCP/IP: Principles, Protocols, and Architecture*, Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Dr. Michael J. Zyda Naval Postgraduate School Code 52, Department of Computer Science Monterey, CA 93943-5100	2
4.	CPT Randolph P. Strong 1514 Camino Way Woodland, CA 95695	2
5.	Capt. Michael C. Winn 4218 N. Barr Oklahoma City, OK 73122	2
6.	Commandant of the Marine Corps Code TE 06 Headquarters, United States Marine Corps Washington, D.C. 20380-0001	1
7.	Mr. Mike Tedeschi United States Army Combat Developments Experimentation Center Attention: ATEC-D Fort Ord, CA 93941	5

Thesis
S8535 Strong
c.1 The Moving Platform Simulator II : a networked real-time visual simulator with distributed processing and line-of-sight displays.

1 FEB 91 36331

Thesis
S8535 Strong
c.1 The Moving Platform Simulator II : a networked real-time visual simulator with distributed processing and line-of-sight displays.

AMERICAN BINDERIES

thesS8535
The Moving Platform Simulator II : a net
3 2768 000 83667 0
DUDLEY KNOX LIBRARY